TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MARKUS RAUTIO

RAVINTOLAN DIGITAALINEN TILAUSJÄRJESTELMÄ

Diplomityö

Tarkastaja: professori Karri Palovuori
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa
31. toukokuuta 2017

TIIVISTELMÄ

MARKUS RAUTIO: Ravintolan digitaalinen tilausjärjestelmä
Tampereen teknillinen yliopisto
Diplomityö, 56 sivua
Elokuu 2017
Sähkötekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Elektroniikka
Tarkastaja: professori Karri Palovuori

Avainsanat: ravintola, digitalisaatio, mobiilisovellus, REST-rajapinta, push-ilmoitukset

Culinar Oy aloitti liiketoimintansa kehittämällä puhtaasti verkkosivuilla toimivan sovelluksen, jota se tarjosi ravintoloille palveluna. Ravintolan asiakas pystyi ravintolan omilta verkkosivuilta tekemään Culinarin sovelluksen avulla ruokatilauksen, jonka ravintola vastaanotti siihen tarkoitukseen tehdyltä verkkosivulta. Verkkosivu tilausten vastaanottamisen välineenä oli kuitenkin ongelmallinen. Ravintoloilta puuttui laite, jonka olisi saanut sovitettua ravintolaan järkevästi ja verkkosivu käyttöympäristönä ei taannut parasta mahdollista käyttökokemusta ja oli altis ongelmatilanteille.

Diplomityössä ratkaistiin verkkosivun ongelma kehittämällä verkkosovelluksen kaltainen, tabletilla toimiva mobiilisovellus. Mobiilisovellus tarjoaa ravintolalle paremman käyttökokemuksen ja Culinar Oy:lle paremmat mahdollisuudet hallita ja monitoroida sovelluksen tilaa etänä. Lisäksi tabletti Culinar Oy:n tarjoamana päätelaitteena sopii hyvin ravintolan hektiseen ja ahtaaseen käyttöympäristöön.

Lähtökohtaisesti työ koostui olemassa olevan verkkosovelluksen ominaisuuksien siirtämisestä sovelluksen muotoon. Erityisesti tilauksen lähettäminen tabletille muuttui siirryttäessä käyttämään mobiililaitteiden omaa push-ilmoitus teknologiaa. Tämän toteuttamiseksi Culinarin koko verkkoarkkitehtuuria kuitenkin piti uudistaa, jotta uusi sovellus kykeni keskustelemaan Culinarin palvelimien kanssa. Työn edetessä ruoantilausprosessia piti myös parantaa yleisemmällä tasolla kehittämällä uusi käytäntö tilauksille, joita ravintola ei hyväksynyt ajallaan.

Työn kehittäminen aloitettiin alkusyksystä 2015 ja saman vuoden alkutalvesta ravintoloilla oli jo käytössä ensimmäinen versio sovelluksesta. Sovellusta kehitettiin aktiivisesti kesään 2016 asti ja tältä väliltä on kerätty dataa sovelluksen toiminnasta ravintolan käytössä. Sovelluksen käyttöönoton ja sen kehittämisen aikana ravintolaan sisään tulevat tilausmäärät kasvoivat ja ravintoloiden vaste tilausten hyväksymiseen pieneni. Hylättyjen tilausten määrä suhteessa hyväksyttyihin tilauksiin pysyi lähestulkoon vakiona koko tämän ajan.

Työ onnistui sille asetetuista tavoitteista teknisestä näkökulmasta erinomaisesti. Työn valmistuminen ei kuitenkaan tarjonnut tarpeeksi liiketaloudellista etua Culinar Oy:lle, minkä vuoksi sen aktiiviselle jatkokehittämiselle ei ole tarvetta tulevaisuudessa.

ABSTRACT

MARKUS RAUTIO: Digital Ordering System for Restaurants
Tampere University of Technology
Master of Science Thesis, 56 pages
August 2017
Master's Degree Programme in Electrical Engineering
Major: Electronics
Examiner: Professor Karri Palovuori

Keywords: restaurant, digitalization, mobile application, REST interface, push notifications

Culinar Oy started its business by developing a purely web based application, which was offered to restaurants as a service. A customer of a restaurant could order food from a web application made by Culinar Oy that was placed in the restaurant's own web page, and the order was received through another web page. However, the web page turned out to be a problematic way to receive orders. The restaurants lacked a device that could be fitted inside the restaurant in a practical way and the user experience of a web page was not good and was prone to errors.

In this thesis, the problem of the web page was solved by creating a tablet based mobile application. The mobile application gives a more tight and controlled user experience while also giving Culinar Oy better possibilities to manage and monitor the application from far away. A tablet as a device can also be placed in the hectic and crowded environment of a restaurant without being a burden.

The first big part of the thesis was to port the existing web application to a mobile application format. One of the biggest changes that were made was how the orders are delivered to the tablet. The mobile application uses a specific way of sending information to it, called the push notifications. To get the new application working with the existing Culinar infrastructure, the interface to the backend server had to be renewed. Also, as the thesis progressed, it became apparent that the process of handling orders that weren't accepted by the restaurant had to be worked on.

The work on the thesis began at the start of fall 2015 and the first versions of the application were in use at the start of winter. The application was actively developed until the summer of 2016 and data about the application's success was gathered during the active development. As the application was launched and improved, the amount of orders the restaurants received grew. Also, the delay of accepting the orders was reduced. The ratio of rejected orders to accepted orders stayed roughly the same throughout the whole time.

The thesis succeeded from a technical perspective and can do well what it was made to do. However, the work did not offer enough economical advantage to Culinar Oy and for this reason, there is no prospects to continue developing the application.

ALKUSANAT

Tämä diplomityö on tehty Culinar Oy:lle, jonka perustin Reetu Kainulaisen ja Jaakko Pasasen kanssa parantamaan ravintoloiden tarjoamaa palvelua. Heille kuuluu suuri kiitos siitä, että he ovat olleet hyviä työkavereita ja ystäviä diplomityöhön liittyvän projektin ja koko Culinarin olemassaolon ajan.

Lisäksi haluan kiittää tarkastajaani Karri Palovuorta, joka on positiivisuudellaan luonut minulle uskoa tämän diplomityön tekemiseen.

Tampereella, 1.8.2017

Markus Rautio

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

3G	engl. Third Generation, kolmannen sukupolven langaton laajaverkkoteknologia
API	engl. Application Programming Interface, ohjelmointirajapinta
APK	engl. Android Package Kit, tiedostoformaatti android-sovelluksille
APNs	engl. Apple Push Notification service, Applen alusta push-ilmoistusen lähettämiselle
CORS	engl. Cross-origin resource sharing, mekanismi rajoitettujen resurssien jakamiseen
CSS	engl. Cascading Style Sheets, kieli merkintäkielien visuaalisen ulkomuodon kuvaamiseen
GCM	engl. Google Cloud Messaging, Googlen alusta push-ilmoitusten lähettämiselle
HTML	engl. Hypertext Markup Language, verkkosivujen luomisessa käytettävä merkintäkieli
HTTP	engl. Hypertext Transfer Protocol, tiedonsiirron protokolla selaimille ja verkkopalvelimille
HTTPS	engl. Hypertext Transfer Protocol Secure, HTTP:n salattu versio
JDK	engl. Java Development Kit, SDK Java-kielelle
JSON	engl. JavaScript Object Notation, tiedostomuoto tiedonvälitykseen
MPNS	engl. Microsoft Push Notification Service, Microsoftin alusta push-ilmoitusten lähettämiselle
REST	engl. Representational State Transfer, HTTP-protokollaan perustuva arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen
SDK	engl. Software Development Kit, joukko työkaluja ohjelmistokehitykseen
SOA	engl. Service-oriented architecture, arkkitehtuuritason suunnittelutapa tietojärjestelmien toimintojen ja prosessien itsenäiseen toteuttamiseen
SOAP	engl. Simple Object Access Protocol, tietoliikenneprotokolla
SoC	engl. Separation of Concerns, suunnitteluperiaate tietokoneohelmiin
SPA	engl. Single Page Application, verkkosovellustyyppi, jossa sivua ei tarvitse ladata uudestaan haettaessa resursseja
SSID	engl. Service Set Identifier, tunniste WLAN:lle
SSL	engl. Secure Sockets Layer, tietoverkkosalausprotokolla
TLS	engl. Transport Layer Security, SSL:n uudempi versio
URI	engl. Uniform Resource Identifier, osoite resurssiin
URL	engl. Uniform Resource Locator, verkkosivun osoite
URN	engl. Uniform Resource Name, resurssin nimi
USB	engl. Universal Serial Bus, sarjaväyläarkkitehtuuri oheislaitteiden liittämiseksi tietokoneeseen
UUID	engl. Universally Unique Identifier, ohjelmoinnissa käytetty tunniste
WLAN	engl. Wireless Local Area Network, langaton lähiverkko
XML	engl. Extensible Markup Language, tiedostomuoto tiedonvälitykseen
XMPP	engl. Extensible Messaging and Presence Protocol, pikaviestinnän ja läsnäolon seurantaan kehitetty teknologia

1. JOHDANTO

Ravintola-ala on teknologisesti kehittyvässä maailmassa tähän asti onnistunut välttämään suuremman digitaalisen murroksen. Ruoka itsessään on fyysinen tuote, jonka vuoksi ravintola-alan ydin liiketoimintaa on mahdotonta muuttaa digitaaliseen muotoon, niin kuin esimerkiksi musiikille on tehty. Ruoan ympärillä toimivat prosessit, kuten ruoan tilaaminen ja kuljettaminen on kuitenkin mahdollista uudistaa tietotekniikan avulla. Culinar Oy perustettiin tarjoamaan ravintolan asiakkaille kanava tilata ruokaa internetin välityksellä. Alkuperäinen tavoite oli tehdä yksi yhteinen verkkosivu, joka kerää eri ravintolat yhteen, ja josta ravintoloiden asiakkaat voivat helposti tilata haluamansa ruoat. Tämä ajatus törmäsi nopeasti kaksisuuntaisten markkinoiden ongelmaan: ravintoloiden asiakkaat eivät käytä verkkosivua, jos siellä ei ole suurta valikoimaa ravintoloita, ja ravintolat eivät halua liittyä sivustoon, jossa ei ole paljon käyttäjiä. Tämänkaltaisen kauppapaikan käynnistäminen vaatii suuren määrän rahaa, mitä pienellä kasvuyrityksellä ei ole. Liikeidea siirtyi malliin, jossa Culinar tarjoaa ravintoloiden omille verkkosivuille palvelun, josta ravintolan olemassa olevat asiakkaat voivat tilata ruoan helposti ja vaivattomasti. Nyt ravintolan asiakkaiden ei tarvitse omaksua uutta toimintamallia uuden verkkosivun käyttämisestä, vaan asiakkaat löytävät palvelun helposti paikasta, jossa he jo valmiiksi käyvät hakemassa tietoa ravintolasta. Tässä mallissa Culinarin palvelu koostuu kahdesta osasta: Ravintolan asiakkaalle näkyvästä, ravintolan verkkosivulla olevasta tilauspalvelusta, sekä ravintolan puolella toimivasta tilausten vastaanottamiseen kehitetystä palvelusta.

Palvelun kehittäminen aloitettiin kesällä 2014 ja saman vuoden joulukuussa palvelun ensimmäinen versio oli käytössä yhdessä ravintolassa. Tässä vaiheessa palvelu oli puhtaasti verkkosivupohjainen. Sekä asiakkaiden, että ravintoloiden näkymä palvelusta oli toteutettu omana verkkosivuna. Palvelu toimi huonosti ja epävarmasti, eikä se saanut tuulta allensa. 2015 vuoden kesään asti palvelua kehitettiin ensimmäisestä pilotista opittujen asioiden pohjalta, ja kesällä 2015 palveluun alkoi saapua uusia ravintoloita. Ravintoloiden lukumäärän kasvaessa palvelussa huomattiin uusi heikkous: Uusien ravintoloiden käyttöönottamisen, ja palvelun tehokkaan toimimisen kannalta tilausten vastaanottaminen verkkosivulla todettiin huonoksi toimintamalliksi. Culinarin seuraavaksi kehityskohteeksi, ja tämän Diplomityön aiheeksi muodostui toteuttaa mobiililaitteella toimiva sovellus, jolla ravintola voi vastaanottaa tilauksia aiempaa tehokkaammin ja luotettavammin. Työn tekninen onnistuminen yksinään ei kuitenkaan riitä tarkasteltaessa lopullista kokonaisuutta, sillä työn kehittämisen motivaatio Culinarin näkökulmasta on sen vaikutus Culinarin taloudelliseen tilanteeseen. Työn todellinen

tavoite on siis parantaa Culinarin edellytyksiä oman liiketoiminnan jatkamiseen ja laajentamiseen.

2. DIGITALISAATIO

Digitalisaatio -kappaleessa käsittelemme aluksi digitalisaatiota yleisenä ilmiönä ja arvioimme sen vaikutusta ravintola-alaan. Tämän jälkeen siirrymme mobiilisovelluksien kehittämiseen, verkkopalveluihin ja langattomiin verkkoihin teknologioina, jotka mahdollistavat ravintoloiden digitalisaation tämän työn kohdalla.

2.1 Digitalisaatio ilmiönä

Digitalisaatio on ilmiö, jonka seurauksena ihmisten tapa kerätä informaatiota, ostaa tuotteita, kuluttaa palveluja, hoitaa asioita, jakaa kokemuksia ja olla vuorovaikutuksessa muiden kanssa muuttuu. Tämän kuluttajakäyttäytymisen muutoksen vuoksi yritysten on myös pakko uudistaa toimintatapojaan ja osaamistaan pysyäkseen kilpailukykyisenä muuttuvilla markkinoilla. Digitalisaatio ilmenee usein internetpalveluina, kuten verkkosivustona, verkkokauppana tai mobiilisovelluksena, mutta sen pohjimmaisena tavoitteena on uusien asiakkaiden tavoittaminen, palvelun tason parantaminen, sekä toiminnan nopeuttaminen ja tehostaminen. Tämän lisäksi digitalisaatiota voidaan myös käyttää uusien liiketoimintamallien, tuotteiden, palveluiden ja prosessien luomiseen.[1]

Terminä digitalisaatio on noussut ihmisten käyttöön viime vuosina. Esimerkiksi suomen mediassa siitä on alettu puhumaan vuosina 2012-2014. Tästä huolimatta digitalisaatiota ei ole juurikaan pyritty määrittelemään ja muun muassa kielitoimiston sanakirjasta ei vielä löydy digitalisaation käsitettä. Usein se tuleekin ilmi esimerkkien avulla, joissa kerrotaan digitalisaation yksittäistapauksista. Tämän vuoksi työssä ei pyritä antamaan digitalisaatiolle tarkkaa määritelmää, vaan valotamme sen taustoja ja yritämme lisätä ymmärrystä siitä, mistä digitalisaatiossa on kyse.[1]

Digitalisaation esivaiheena voidaan pitää 1990-luvulta asti tunnettua *digitalisoitumista*. Digitalisoitumisessa on kyse analogisten esineiden ja prosessien muuntamisesta digitaliseen muotoon osittain tai kokonaan. Tämä muuntaminen analogisesta digitaaliseen kulkee nimellä *digitalisointi*. Esineiden digitalisoimisesta esimerkkejä ovat analogisten vinyylien vaihtuminen ensin digitaalisiin CD-levyihin, ja CD-levyistä siirtyminen musiikin suoratoistopalveluihin, kuten Spotifyhyn. Toinen esimerkki on valokuvista siirtyminen digikuviin, joista on siirrytty valokuvien pilvipalveluihin. Prosessin digitalisoimisesta esimerkkinä toimii asuntolainahakemus. Entinen paperinen lomake voidaan täyttää nyt sähköisesti verkossa, ja tämän seurauksena itse hakemuksen käsittely voidaan suorittaa sähköisesti tai jopa automaattisesti. Digitalisoitumisesta siirrytään digitalisaatioon, kun se muuttaa ihmisten käyttäytymistä, markkinoiden dynamiikkaa ja yritysten ydintoimintaa. Tämän muutoksen toteuttamisessa hyödynnetään teknologiaa, vaikkakaan teknologia itsessään ei aiheuta digitalisaatiota.[1]

Digitalisaatio voidaan jakaa makro- ja mikrotasoon riippuen siitä, miten asiaa tarkastellaan. Makrotasolla puhutaan koko yhteiskunnan, talouden, markkinoiden ja ihmisten toimintatapojen muutoksista, kun taas mikrotasolla tarkastellaan yksittäisiin yrityksiin kohdistuvista muutoksista. Nämä kaksi tasoa elävät vuorovaikutuksessa toistensa kanssa jatkuvasti. Esimerkiksi makrotasolla tapahtuva säätely vaikuttaa yksittäisten yritysten toimintaan, ja toisaalta yksittäiset yritykset muovaavat markkinoiden rakennetta. On hyvä ymmärtää tämä laajempi konteksti, mutta työssä ei syvennytä yhteiskunnan digitalisaation tämän enempää, vaan keskitymme yrityksen tasolla tapahtuvaan digitalisaatioon.[1]

Yrityksen näkökulmasta kyse on siis liiketoiminnan uudistamisesta, jonka kautta pyritään lisäämään kasvua, kannattavuutta ja kilpailukykyä. Digitalisaatio tarjoaa työkalut, joilla nämä lopputulokset voidaan saavuttaa neljän keinon avulla.

1. Liikevaihdon kasvattamiseen pyritään tavoittamalla uusia markkinoita ja kohderyhmiä ja täten lisäämällä myyntiä. Asiakkaille voidaan tarjota henkilökohtaisia suosituksia ja kohdennettua markkinointia kerätyn datan avulla. Lisäksi uutta ansaintaa voidaan saavuttaa digitaalisilla palveluilla ja liiketoimintamalleilla.
2. Kustannuksia alennetaan tehostamalla toimintaa. Tämä tapahtuu automaation ja itsepalvelun kautta. Myös paperin, postittamisen ja toimitilojen tarve vähenee, kun ihmiset kuluttavat palveluja digitaalisesti.
3. Pääoman käyttöä voidaan tehostaa joko lisäämällä sen kiertonopeutta tai minimoimalla sen tarvetta. Analysoimalla liiketoiminnasta kertyvää dataa, vanhoja toimintamalleja, kuten varaston optimointia, voidaan parantaa. Sähköisen laskutuksen avulla pääoma saadaan liikkumaan välittömästi haluttuna ajankohtana.
4. Uudistamalla liiketoimintaa rakenteellisesti, eli *transformoimalla* sitä, yritys tekee mahdollisesti isoja muutoksia liiketoimintaansa, kuten siirtymällä tuotteiden tarjoamisesta palveluiden tarjoamiseen, tai muuttamalla myyntikanaviaan. Transformaatio tähtää pidempiaikaiseen kasvun, kannattavuuden ja kilpailukyvyn parantamiseen, joka lyhyellä aikavälillä näkyy rakenteellisena muutoksena.

Useat konkreettiset digitalisaation toimenpiteet päätyvät yhdistämään näitä keinoja. Culinarin toteutus on yhdistelmä kustannusten alentamista ja liiketoiminnan transformointia. Erityisenä kohteena on kustannusten alentaminen myynnin tehokkuutta kasvattamalla. Verkossa myyty tuote on ravintolalle halvempi, sillä työntekijän aikaa ei kulu turhaan puhelimen ääressä.[1]

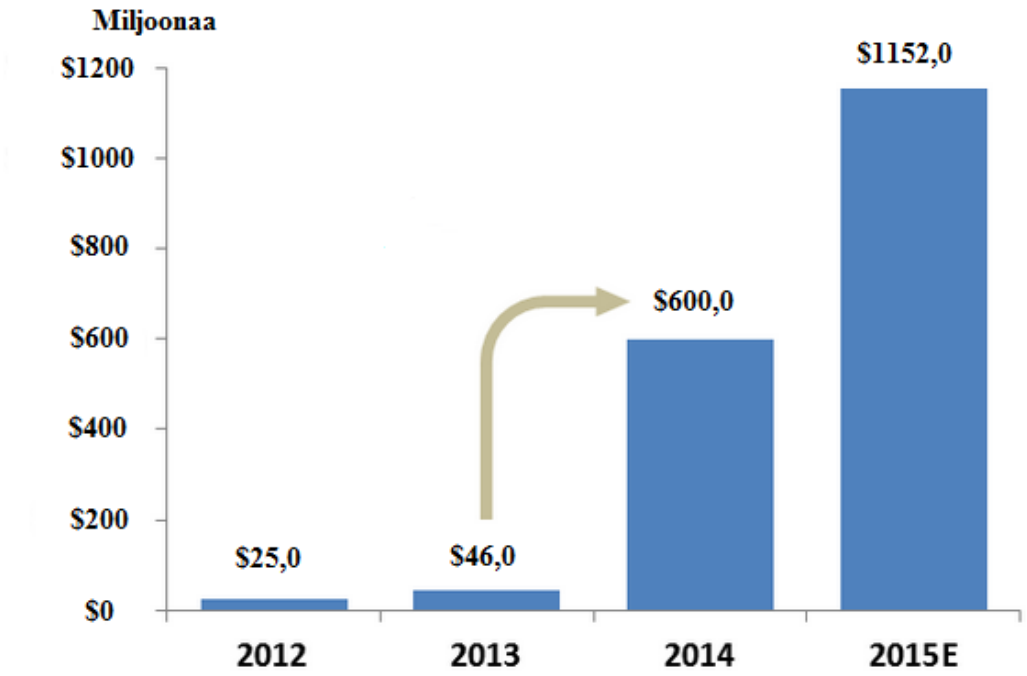
Digitalisaation ajavina voimina toimivat *asiakaskäyttäytymisen*, *teknologian* ja *markkinoiden* murrokset. Asiakaskäyttäytymisen murroksessa asiakkaan tapa toimia on muutoksen alla. Asiakkailla on internetin vuoksi entistä enemmän informaatiota

saatavilla ja globaalien markkinoiden vuoksi hänellä on pääsy yhä laajempaan valikoimaan kilpailevia tuotteita. Älylaitteiden avulla asiakkailla on kyky käyttää palveluita ja tuotteita yhä useammissa konteksteissa ja ympäristöissä. Tästä seuraa, että asiakkaat ovat aiempaa hintatietoisempia ja odottavat parempaa palvelua. Joissakin tilanteissa parempi palvelu tarkoittaa itsepalvelun kasvamista, sillä asiakkaat saavat tehdä kulutuspäätöksiä omalla tahdillaan. Toisaalta palvelua tarvitessa sitä halutaan välittömästi. Informaation lisääntyminen tarkoittaa myös sitä, että tieto tuotteista lisääntyy ja muiden asiakkaiden arviot kyseisistä tuotteista ohjaavat ostokäyttäytymistä entistä enemmän. Teknologian murroksessa kyse on tietotekniikan erittäin nopeasta kehityksestä, joka toimii mahdollistajana digitalisaatiolle. Prosessorien tehokkuuden lisääntyminen ja tiedonsiirron nopeuden kasvu yhdistettynä niiden laskevaan hintaan on tuonut kuluttajille halvan mahdollisuuden kuluttaa uusia ja älykkäämpiä palveluita. Markkinoiden murroksessa yrityksillä on aiempaa suurempi mahdollisuus laajentua uusille markkinoille digitaalisia kanavia hyödyntämällä. Tämä kanavien laajuus on synnyttänyt paljon uusia yrityksiä, jotka ovat aggressiivisesti panostaneet usean eri markkinan yhtäaikaiseen hallintaan. Sen lisäksi, että palveluja tarjotaan uusilla markkinoilla paikallisesti, kyse on myös siitä, että tuotteita toimitetaan maailmanlaajuisesti. Digitalisaation vaikutus näkyy myös siinä, että yritykset laajentuvat toimialoista toiseen. Tästä esimerkkinä Amazon, joka kirjakaupan lisäksi tarjoaa Kindle-lukulaitetta, jolla kirjoja voi lukea sähköisessä muodossa.[1]

2.1.1 Ravintola-alan digitalisaatio

Suomalaisista noin 90 prosenttia käyttää internetiä ja lähes 70 prosentilla on älypuhelin. Älypuhelimesta on tullut digitaalisten palveluiden ja sisältöjen yleisin käyttöväline ja tämä näkyy siinä, että yli puolet suomalaisista käyttävät internetiä kodin ja työpaikan ulkopuolella. Tämän lisäksi verkkokaupassa on asioinut jo yli puolet suomalaisista 16-74-vuotiaista. Verkkokaupan kasvu on ollut noin 5-10 prosentin luokkaa vuosittain, mutta 2014 taloustilanteen vuoksi se oli vain hieman edellisvuotta suurempi. Tämäkin hidastuminen näkyi vain ulkomaanmatkojen myynnin vähenemisenä, sillä tavaroiden myynti kasvoi 7 prosenttia ja digitaalisen sisällön kulutus jopa 12 prosenttia. Verkossa asiointi on siis Suomessa jatkuvassa kasvussa.[1] Culinarin aloittaessa 2014 syksyllä, ravintola-alan verkkomyynti oli kuitenkin täysin olematonta pizzerioihin kohdistuvaa Pizza-online palvelua lukuun ottamatta. Maailmanlaajuisesti ravintola-alan digitalisaatio oli kuitenkin jo lähtenyt valtavaan kasvuun. Kuvasta 2.1 nähdään, että 2014 ruoan online-tilauspalveluita kehittäviin yrityksiin sijoitettiin 600 miljoonaa dollaria. Kyseessä on 13-kertainen kasvu 2013 vuoden 46 miljoonasta dollarista. 2015 toukokuussa ennustettiin 1,152 miljardin dollarin sijoituksia koko vuodelle.[2] Maailmanlaajuisesti kyseessä on siis megatrendi, johon suomessa alettiin herätä hieman muita myöhemmin. Vuoden 2014 marraskuussa järjestetyssä Slush-tapahtumassa suomalainen ruokaa kotiin kuljettava online-tilauspalvelu Wolt julkisti saaneensa 400 000 euron sijoituksen ja aloittavansa

liiketoiminnan vuonna 2015. 2015 kesällä Suomeen rantautui myös kansainvälinen ruokaa kotiin kuljettava tilauspalvelu Foodora.



Kuva 2.1 *Sijoitukset online-tilausyrityksiin*

Maailmanlaajuisesti vuonna 2015 verkossa tehtyjen kotiinkuljetustilausten ja paikan päältä mukaan noudettavien tilausten rahalliseksi arvoksi arvioitiin 9 miljardia dollaria, kun taas verkon ulkopuolella tehtyjen tilausten arvoksi arvioitiin 61 miljardia dollaria. Verkossa tehtyjen tilausten osuus on kuitenkin jatkuvassa kasvussa ja sen on arvioitu ohittavan verkon ulkopuolella tehtyjen tilausten määrän 2022 vuoden jälkeen.[2] Ravintola-alan digitalisaatiossa on siis olemassa valtava liiketoiminnallinen potentiaali, jonka kanssa ollaan kuitenkin monilla markkinoilla alkutekijöissään.

2.2 Mobiilisovelluksien kehittäminen

Tässä kappaleessa käymme läpi mobiilisovelluksien kehittämiseen liittyviä tekijöitä yleisesti ja syvennymme valittuihin yksityiskohtiin, jotka ovat työn kannalta oleellisia.

2.2.1 Mobiilisovelluksien tyypit

Modernien älypuhelimien ja tablettien yleistyttyä mobiilisovelluksien kehittäminen on kasvanut eksponentiaalisella vauhdilla. Tapamme päästä käsiksi informaatioon ja kommunikoida ideoita on kokenut voimakkaan muutoksen mobiilien verkkosovellusten noustua perinteisten käyttöjärjestelmälle kehitettävien natiivien sovellusten rinnalle.[3]

Natiivi mobiilisovellus on älypuhelimen käyttöjärjestelmälle (Android, iOS) kehitetty sovellus, joka käännetään käyttöjärjestelmän kehityskielestä (esimerkiksi Java, Objective-C ja C#) ja allekirjoitetaan käyttäen alustan virallisia kehitystyökaluja. Sovelluksen kehittäjät voivat lisätä allekirjoitetun sovelluksen käyttöjärjestelmän omaan kauppapaikkaan sovelluksen täyttäessä kauppapaikan sovellusehdot. Tämän jälkeen kuluttajat voivat ladata sovelluksen kauppapaikasta ja asentaa sen älylaitteelleen. Sovellus siis sijaitsee älylaitteella ja käynnistettäessä älylaite suorittaa sovellusta laitteistollaan. Tämän vuoksi natiivilla sovelluksella on suora pääsy älylaitteen sisäisiin palveluihin, kuten kameraan tai osoitekirjaan, kehitystyökalujen tarjoamien rajapintojen avulla. Vaikka natiivi sovellus ei itse kykene suorittamaan verkkosivuja, natiivien sovellusalustojen kehitystyökalut tarjoavat komponentin, jolla natiivin sovelluksen sisällä voidaan näyttää verkkomediaa. Tällaista komponenttia kutsutaan *web view*:ksi.[4]

Verkkosovellus on JavaScriptin, Hypertext Markup Languagen (HTML) ja Cascading Style Sheets:n (CSS) avulla rakennettu selaimessa ajettava internetsivu, joka huomioi mobiililaitteiden nykyaikaiset tarpeet. Näillä työkaluilla on mahdollista tehdä verkkosovelluksesta funktionaalisuudeltaan ja käyttäytymiseltään hyvin natiivin sovelluksen kaltainen. Verkkosovellus on esimerkiksi kykenevä mukauttamaan näkymänsä riippuen tarkasteltavan laitteen näytön koosta.[3] Ero perinteisen verkkosivun ja verkkosovelluksen välillä on kuitenkin häilyvä ja sanaa sovellus käytetään juurikin kuvaamaan verkkosivua, joka muistuttaa natiivia sovellusta. Single-page application (SPA), eli yhden sivun sovellus on verkkosovelluksen muoto, joka muistuttaa erityisen paljon natiivia sovellusta. Verkkosivut käyttävät Uniform Resource Locator -osoitteita (URL) sivun eri osioiden välillä navigointiin. URL-osoite on referenssi resurssiin verkossa ja määrittelee sen sijainnin ja hakutavan.[5] Kun URL-osoite syötetään selaimen osoiteriville, selain lähettää kutsun URL-osoitetta vastaavalle palvelimelle, joka palauttaa selaimelle sivun näyttämiseen tarvittavat tiedot. Palvelin muuntaa digitaalisen tiedon näytölle sopivaan esitysmuotoon, eli *renderöi* verkkosivun. Perinteisen verkkosivun sisällä sivulta toiselle navigointi tapahtuu käyttäen hyperlinkkejä, jotka muuttavat sivun URL:n, hakevat palvelimelta uuden sivun tarvitsemat tiedot ja suorittaa koko sivun renderöinnin uusiksi. SPA sovellus hakee palvelimelta välittömästi eri sivujen näyttämiseen tarvittavat tiedot. Tämän jälkeen sovelluksessa navigoitaessa sivulta toiselle selain renderöi vain muuttuvan informaation ohjelmallisesti, mutta ei hae tietoa palvelimelta tai renderöi koko sivua uudelleen.[6] Vain dynaaminen, alati muuttuva informaatio haetaan palvelimelta Ajax-kutsujen avulla. Ajax termiä käytetään kuvaamaan verkkosovelluskehityksen tekniikoita, jotka mahdollistavat asiakasohjelman ja palvelimen välisen tiedonvälityksen ilman koko sivun uudelleenlataamista.

Verkkoteknologioiden kehittyessä kiinnostus verkkosovellusten kehittämiseen on lisääntynyt. Koska verkkosovellus suoritetaan käyttäen jokaisessa selaimessa toimivia teknologioita, voidaan se ajaa jokaisessa selaimessa samalla koodikannalla, kunhan selainten uniikit eroavuudet on huomioitu ohjelmassa. Verkkosovellus on siis

käytettävissä jokaisella älylaitteella ja vaatii vain yhden koodikannan ylläpitoa. Natiivi sovellus pitää kirjoittaa jokaiselle alustalle erikseen omalla kielellään. Universaalin saatavuuden takaaminen vaatii siis enemmän työtä, jonka vuoksi se on myös kalliimpaa. Useat sovellukset kuitenkin vaativat pääsyn älylaitteen laitteistoon tai sen omaan ohjelmistoon toimiakseen tarkoitetulla tavalla.[3] Näiden kahden sovellustyypin hyötypuolia on pyritty tuomaan yhteen *hybridisovellusten* mallilla.

Hybridisovellus, tai natiivi verkkosovellus, on käännetty natiivi sovellus, joka on paketoitu web view:n sisälle. Koska web view kykenee esittämään verkkomediaa, voidaan sovellus kehittää käyttäen perinteisiä verkkoteknologioita (CSS, HTML, JavaScript), mikä tarkoittaa yhtä koodikantaa kehittäjälle. Hybridisovelluksessa yhteensopivuus web view:n ja älylaitteen alustan välillä on ulkoistettu kolmannelle osapuolelle, kuten *Apache Cordova*:lle, jonka avulla hybridisovellus myös pääsee käsiksi käyttöjärjestelmän natiiveihin rajapintoihin.

2.2.2 Push-teknologia

Monet nykyaikaiset älypuhelinsovellukset pohjautuvat jatkuvaan ja reaaliaikaiseen tiedonvälitykseen. Tämän suuntauksen tukemiseksi mobiilikäyttöjärjestelmien kehittäjät ovat rakentaneet push-ilmoitusten infrastruktuurin, jonka avulla kolmannen osapuolen kehittäjät voivat helposti hallita puhelimelle lähetettäviä viestejä. Push-palvelut huolehtivat siitä, että päätelaitteelle lähetetyt viestit pääsevät perille laitteeseen asti.[7]

Push-arkkitehtuurin voi teknisesti toteuttaa muutamalla erilaisella tavalla. Jokaisessa toteutuksessa on kuitenkin yhteisiä tekijöitä. *Yhdyskäytävä* on vastuussa sisällön lähettämisestä *asiakaslaitteille* ilmoituskanavia pitkin. Lähetetty sisältö syntyy *sisällöntuottajalla*, joka on myös vastuussa sovelluksen palvelinpuolesta ja kommunikoi tilan muutoksista asiakasohjelmalle. Sisällöntuottaja aloittaa ilmoitusprosessin tekemällä pyynnön yhdyskäytävälle. Yhdyskäytävä piilottaa taaksensa asiakaslaitteen kanssa käytävän kommunikaation monimutkaisuuden ja tarjoaa väylän ajankohtaiseen ja asynkroniseen viestintään. Tässä mallissa luotettavuus, tehokkuus, turvallisuus ja skaalautuvuus nousevat tärkeiksi tekijöiksi.[8]

Luotettavuus on erityisen suuri tekijä, kun huomioidaan älylaitteille ominainen jatkuva verkkoon kytkeytyminen ja verkosta irtautuminen. Laite voi irrota verkosta tyhjentyneen akun vuoksi, ollessa katvealueella tai koska sen yhteydet on suljettu manuaalisesti. Toteutetussa push-palvelussa itsessään voi myös olla luotettavuuteen liittyviä, päätelaitteesta riippumattomia ongelmia. *Tehokkuutta* voidaan tarkastella useammasta näkökulmasta. Yhdyskäytävän saadessa push-pyynnön, sen tulee toimittaa ilmoitus mahdollisimman pienellä viiveellä. Älylaitteillä on usein käytössä rajallinen kaistanleveys, jonka vuoksi push-palvelun tulee minimoida kaistankäyttö. Push-protokollan tulee olla myös tehokas energiankäytöltään laitteiden rajallisten akkujen vuoksi. *Turvallisuus* on monelle sovelluksen kehittäjälle tärkeää. Luottamuksellisuuden

ja ilmoitusten eheyden tarjoamiseksi push-palvelun pitää turvata sisällöntuottajan yhteys yhdyskäytävään, sekä yhdyskäytävän yhteys asiakasohjelmaan. *Skaalautuvuus* kertoo yhdyskäytävän kyvystä palvella kasvavaa sisällöntuottajien ja asiakasohjelmien määrää ja lisääntyvää push-ilmoitusten liikennettä. Yhdyskäytävän tulee myös hallita arvaamattomia push-pyyntöjen piikkejä.[8]

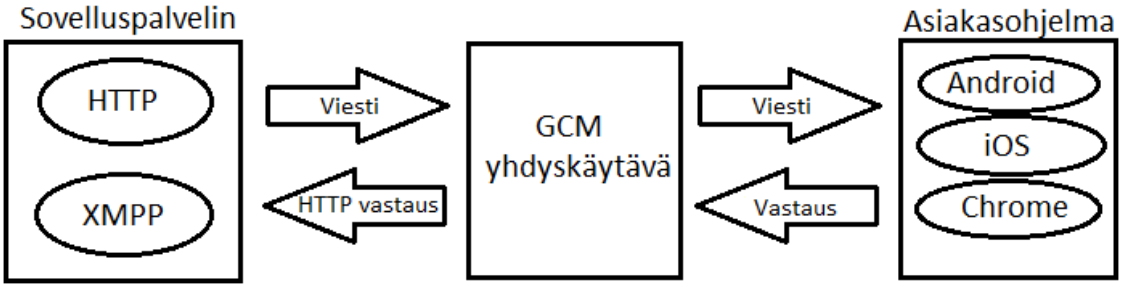
Push-ilmoitus on tapahtumapohjainen mekanismi, jossa etäpalvelin lähettää tapahtuman asiakasohjelmalle kyseisen tapahtuman sattuessa (esimerkiksi lähetettäessä pikaviesti). Koska älylaitteen pääasiallinen keino tiedonvälitykseen on matkapuhelindataverkot, yhdyskäytävää käytettäessä laitteiden tavoitettavuudessa kohdataan kaksi ongelmaa:

- Osoitettavuuden ongelmassa lähetettävän ilmoituksen päätepiste on hankalasti määriteltävissä. Matkapuhelinverkkojen operaattorit tyypillisesti piilottavat päätelaitteiden osoitteet verkossa, jonka vuoksi ei tiedetä minne ilmoitus tulisi lähettää.
- Päätelaitteen saavutettavuudessa kyse on ilmoituksen reitittämisestä laitteelle sen liikkuessa dynaamisesti eri verkoissa. Operaattorit yleisesti estävät palomuureilla niiden verkkojen ulkopuolelta tulevan liikenteen.

Tavoitettavuuden ongelma voitaisiin ratkaista kysymällä jatkuvasti palvelimelta (pollaamalla) onko sovelluksen kannalta oleelliset tiedot muuttuneet. Pollaaminen ei kuitenkaan ole tehokas tapa ratkaista tätä ongelmaa. Jos palvelinta pollataan useammin kuin oleellinen tieto muuttuu, käytetään turhaan kaistanleveyttä, sekä laitteen akkua aktivoitaessa sen radiovastaanotinta. Jos palvelinta pollataan liian harvoin, palvelimella tapahtuneita muutoksia ei saada ajallaan. Pollaaminen aiheuttaa myös palvelimen infrastruktuurille skaalautuvuuden ongelman asiakasohjelmilta tulevan suuren liikenteen vuoksi. Push-ilmoitukset pyrkivät mahdollisimman suureen tehokkuuteen uhraamatta tiedon tuoreutta. Asiakasohjelma muodostaa yhteyden palvelimeen, ja lähettää jaksollisesti sydämenlyönti-viestejä yhteyden ylläpitämiseen. Jos jommankumman puolen sydämenlyönti huomaa yhteydessä ongelman, yhteys pyritään muodostamaan uudelleen. Tarkoituksena on lähettää sydämenlyöntejä minimaalinen, mutta tarpeellinen määrä yhteyden ylläpitämiseksi.[8]

Käytännössä jokaisella mobiilikäyttöjärjestelmällä on oma palvelunsa ilmoituksille. Googlen Android käyttää Google Cloud Messagingia (GCM), Applen iOS käyttää Apple Push Notification Serviceä (APNs), Microsoftin Windows Phone käyttää Microsoft Push Notification Serviceä (MPNS) ja niin edelleen. Vaikka näiden palveluiden ominaisuuksissa ja teknisissä toteutuksissa on eroja, tietyn käyttöjärjestelmän laitteet ovat pitkälti rajoittuneita oman käyttöjärjestelmänsä sisälle. Poikkeuksena toimii GCM, joka toimii myös Applen iOS ympäristössä. Tästä johtuen eri palveluiden vertaileminen tarkemmin teknisellä tasolla ei ole tässä työssä oleellista, koska ilmoituspalvelua on lähes mahdotonta valita vapaasti. Tämän lisäksi sovellukset, jotka haluavat toimia useammassa kuin yhdessä mobiilikäyttöjärjestelmässä, joutuvat ottamaan käyttöön useamman

ilmoituspalvelun hyötyineen ja haittoineen riippumatta niiden eroavaisuuksista. Koska työssä päädyttiin käyttämään GCM:ää, sen toiminnasta kerrotaan tarkemmin.



Kuva 2.2 *GCM-arkkitehtuuri*

Google Cloud Messaging on ilmainen Googlen palvelu Android alustalle. Sen avulla palveluntarjoaja voi lähettää asiakasohjelmalle maksimissaan 4 kilotavun kokoisia ilmoituksia.[9] Kuten kuvasta 2.2 nähdään, GCM:n yhdyskäytävänä toimiva palvelin hyväksyy sovelluspalvelimelta tulevat viestit ja lähettää ne asiakasohjelmalle. Tätä varten asiakasohjelman pitää rekisteröityä GCM:n yhdyskäytäväpalvelimella saadakseen uniikin rekisteröitymispoletin, jonka se antaa sovelluspalvelimen käyttöön. Kun sovelluspalvelimelle tulee tarve lähettää viesti asiakasohjelmalle, se lähettää viestin ja asiakasohjelman poletin yhdyskäytävälle hypertekstin siirtoprotokollaa (HTTP, engl. Hypertext Transfer Protocol) ja/tai Extensible Messaging and Presence Protocol:a (XMPP) hyödyntäen. Yhdyskäytävä varastoi sovelluspalvelimelta tulevat viestit ja lähettää ne polettia vastaavalle asiakasohjelmalle. Jos asiakasohjelma haluaa lopettaa viestien vastaanottamisen, voi se perua rekisteröitymisensä yhdyskäytävälle, jolloin vanha poletti lakkaa toimimasta. Asiakasohjelma voi milloin tahansa pyytää uuden poletin yhdyskäytävältä sen halutessaan.[9]

2.3 Verkkopalvelut

Verkkopalvelu on nimensä mukaisesti palvelu, jonka avulla verkon yli siirretään dataa ja informaatiota. Verkkopalvelut toteuttavat palvelulähtöistä arkkitehtuuria (SOA, engl, Service-oriented architecture). SOA:n ajatus on se, että verkkopalvelu on käytettävissä useassa eri applikaatiossa riippumatta siitä, mitä teknologiaa tai alustaa käytetään. SOA tarjoaa siis yhtenäistetyn tavan kommunikoida halutun palvelun kanssa.[10] Käytännössä on olemassa kaksi erilaista tapaa rakentaa SOA:n mukainen verkkopalvelu: Simple Object Access Protocol (SOAP) ja Representational State Transfer (REST). SOAP on Extensible Markup Language:lla (XML) kirjoitettujen dokumenttien vaihtamiseen tarkoitettu protokolla, jossa keskustelu verkkopalvelun kanssa voidaan toteuttaa useamman eri standardin välityksellä. Tästä johtuen SOAP-palveluiden kehittäminen vaatii erilaisten standardien ymmärtämistä ja oikeanlaisten työkalujen valitsemista vastaamaan näitä standardeja. SOAP määrittelee asiakasohjelman ja verkkopalvelun välille tiukan kommunikaatioon ja toimintoihin pohjautuvan keskustelun

protokollan.[11] Työssä käytetään REST-mallia, jonka vuoksi SOAP:n ei mennä tämän syvemmälle.

REST on resurssipohjainen HTTP-protokollaa hyödyntävä tilaton verkkopalvelu. Palvelun tilattomuus tarkoittaa sitä, että palvelu itsessään ei ylläpidä tietoa asiakasohjelman tilasta, vaan asiakasohjelman kommunikoidessa palvelimen kanssa, se lähettää tarvittavat tiedot aina palvelimelle. Ennen kuin kerromme tarkemmin miten REST toimii, käymme läpi sen ymmärtämisen kannalta oleelliset asiat, joiden päälle REST rakentuu.

2.3.1 Uniform Resource Identifier

Uniform Resource Identifier (URI) on merkkijono, jolla identifioidaan resurssi. Selkeyden vuoksi URI jaetaan usein kahteen alakategoriaan: Uniform Resource Locatoriin (URL) ja Uniform Resource Nameen (URN). URL kertoo resurssin osoitteen palvelussa, minkä vuoksi URL:ä kutsutaan usein verkko-osoitteeksi. URN taas kertoo resurssin tarkan nimen. Esimerkkinä URN:stä on kirjan tai muun erillisteoksen kansainvälinen standarditunnus ISBN, joka määrittelee kaikille teoksille uniikin numerosarjan, jolla ne tunnistetaan. Koska pelkkä resurssin tunnus ei kerro kuinka tämä resurssi voidaan hakea, se ei ole hyödyllinen työkalu verkkopalvelujen rakentamiseen. REST hyödyntää siis URL:ä resurssiensa identifioimiseen. Vaikka virallisten standardien mukaan URL on vain hyödyllinen termi kuvaamaan tietynlaista URI tyyppiä, sen merkittävyyden vuoksi puhumme jatkossa myös paljon URL:sta.[12]

URI:n yleinen syntaksi muodostuu seuraavasti:

skeema:[//[käyttäjä:salasana@]isäntä[:portti]][/]polku[?kysely][#sirpale] [13]

Skeema kertoo yleisen tiedon tyypin, joka voi myös kertoa käytetyn protokollan. Tällaisia protokollia ovat esimerkiksi HTTP ja File Transfer Protocol, mutta yleisemmin skeema voi olla mailto sähköpostiosoitteen yhteydessä tai file, jos haetaan tiedostoa paikallisen tietokoneen tiedostoavaruudesta. Koska REST pohjautuu HTTP-protokollaan, REST:ssä skeema on aina joko HTTP, tai HTTP-protokollan salattu versio Hypertext Transfer Protocol Secure (HTTPS). HTTP:n ja HTTPS:n yhteydessä URI:sta käytetään yleisesti termiä URL. URI:ssa voi olla sisällytettynä autentikointi-osuus, jolloin skeeman jälkeen kerrotaan käyttäjätunnus ja salasana. Tämä ei ole yleinen käytäntö verkkopalvelun kanssa, vaan autentikointi suoritetaan jollakin muulla, verkkopalvelun erikseen määrittelemällä tavalla. URI:ssa esiintyvä autentikointi voi olla käytössä esimerkiksi otettaessa yhteyttä suoraan tietokantaan, jolloin muunlaista autentikointia ei ole mahdollista käyttää. URI:n isäntä osuus on joko rekisteröity isäntänimi tai IP-osoite, jonka avulla asiakasohjelma löytää palvelua tarjoavan palvelimen sijainnin verkossa. Portti on numero, joka kertoo mihin palvelimen porttiin yhteys ollaan muodostamassa. HTTP-protokollan oletusportti on 80 ja HTTPS-protokollan 443, mutta palvelun tarjoaja

voi itse päättää mitä porttia käyttää. Polku on palveluntarjoajan määrittelemä polku kyseisellä palvelimella resurssin sijaintiin. Polku voi olla absoluuttinen polku palvelimen tiedostorakenteessa, tai mielivaltaisesti päätetty polku. Polku on aina määritelty ja alkaa aina kauttaviiva merkillä (/), vaikka polku jätettäisiinkin tyhjäksi. URI:n kysely osuus aloitetaan kysymysmerkillä (?) ja määrittelee parametrit, jotka vaikuttavat resurssin hakemiseen. URI:n sirpale osuus aloitetaan ristikkomerkillä (#). Tässä osiossa määritellään toissijaisia parametreja, jotka esimerkiksi kertovat mistä kohdasta haettu resurssi näytetään.[5; 13]

Todellinen URI verkkopalvelussa voi esimerkiksi olla seuraavan lainen:

https://www.esimerkki.com/kaverit/lista?kayttajaId=123456

Kyseessä on siis HTTPS-protokollaa hyödyntävä, www.esimerkki.com isäntänimen takana oleva palvelin. Koska porttia ei ole erikseen määritelty ja protokollana on HTTPS, käytetään oletusporttia 443. Polku määrittelee, että haettava resurssi on lista kavereista. Lopuksi on vielä kyselyparametriksi annettu käyttäjän id, jolloin palvelin saa tiedon siitä, kenen kaverilistasta on kyse. Esimerkin URI:n muoto on varsin yleinen REST verkkopalveluissa, ja antaa hyvän käsityksen siitä millaiselta REST palveluun tehtävät kutsut näyttävät.

2.3.2 Hypertext Transfer Protocol

Hypertext Transfer Protocol on sovelluskerroksen protokolla, jota käytetään tiedonsiirtoon selaimien ja/tai verkkopalveluiden välillä. World Wide Web:n informaation vaihto rakentuu HTTP:n päälle ja World Wide Web:llä tarkoitetaankin nimenomaan Internetissä toimivia verkkoja, jotka keskustelevat HTTP:n välityksellä. Tämä on helppo sotkea Internetiin, joka on laajempi kokonaisuus ja käsittää muita tiedonsiirron protokollia, muun muassa sähköpostin. Hyperteksti on strukturoitua, loogisista linkeistä(hyperlinkeistä) koostuvaa tekstiä. Hyperlinkit taas ovat viittauksia toisiin dokumentteihin, jolloin hyperteksti muodostuu useista dokumenttien ristiviittauksista. HTTP protokollana määrittelee, kuinka hypertekstiä vaihdetaan ja siirretään.[13]

HTTP toimii pohjautuen pyyntö-vastaus metodiin asiakas-palvelin tietojenkäsittelymallissa. Tämä tarkoittaa sitä, että esimerkiksi verkkoselain voi toimia asiakkaana ja tietokoneella verkkosivua tarjoava sovellus palvelimena. Asiakas lähettää palvelimelle HTTP-pyynnön, johon palvelin vastaa lähettämällä asiakkaalle pyynnön mukaisesti informaatiota, joka voi olla HTML-tiedostoja tai muuta sisältöä. Palvelin voi myös suorittaa muuta toiminnallisuutta asiakkaan puolesta, jolloin se vastaa vain kertomalla kyseisen toiminnallisuuden suoriutumisesta. HTTP määrittelee pyynnölle metodeja, jotka vaikuttavat palvelimen tapaan reagoida pyyntöön. Nämä metodit ovat GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE ja CONNECT. Kun tiettyyn

URL:n ollaan tekemässä HTTP-pyyntö, edellä mainitut metodit määrittävät mitä URL:n osoittamalle resurssille ollaan tekemässä. Metodien havainnollistamiseksi otamme kuvitteellisen URL:n http://esimerkki.com/kirja/kappale, johon viittaamme seuraavan kappaleen aikana.[14]

GET metodi pyytää resurssin representaation. GET pyyntöjen tulisi vain hakea informaatiota eikä sillä tulisi olla muuta vaikutusta. Esimerkki URL:n tehty GET pyyntö hakisi kirjan kappaleen, ja jotta palvelin tietää mistä kirjasta ja kappaleesta on kyse, URL:ssa kulkisi myös kyselyosiossa parametreina kirjan ja kappaleen tunnisteet. HEAD metodi hakee GET metodin tavoin informaatiota palvelimelta, mutta ilman vastauksen runkoa. Tämä on hyödyllistä, jos tarvitsee hakea vain meta-tietoa, jota on saatavilla vastauksen ylätunnisteissa. POST metodi pyytää palvelinta lisäämään pyynnön rungossa olevan entiteetin URL:n määrittelemän resurssin instanssiksi. Toisin kuin GET pyynnössä, jossa palvelimen tarvitsemat tiedot kulkevat mukana URL:ssä, POST pyynnössä on mukana runko, johon voi laittaa enemmän informaatiota. Esimerkki URL:n tehty POST pyynnön rungossa kulkisi kirjan kappale, jonka palvelin lisää uudeksi kirjan kappaleeksi. Kappaleen lisäksi rungossa tulisi siis myös kulkea tieto siitä, mihin kirjaan kappale ollaan lisäämässä. PUT metodi on hieman kuten POST metodi, mutta verrattuna POST metodin uuden instanssin luomiseen, PUT metodi ensisijaisesti korvaa jo olemassa olevan resurssin instanssin rungossa olevalla tiedolla, ja jos päivitettävää instanssia ei ole olemassa, se luo uuden instanssin annetuilla tiedoilla. Esimerkki URL:n tehty PUT metodi siis yrittäisi ensin päivittää jotain tiettyä kirjan kappaletta, ja jos kappaletta ei löydy, se luo kirjaan uuden kappaleen. DELETE metodi poistaa URL:n kyselyosiossa määritellyn instanssin, mikä tarkoittaa esimerkissä tietyn kirjan kappaleen poistamista. TRACE metodissa palvelin vain palauttaa välittömästi lähetetyn pyynnön tekemättä mitään muuta, jolloin on mahdollista tarkastella mitä pyynnölle tapahtuu palvelimilla, joiden läpi pyyntö kulkee lopulliselle palvelimelle päästäkseen. Tätä metodia voi käyttää esimerkiksi testi- ja diagnostiikka tarkoitukseen. OPTIONS metodi palauttaa palvelun tukemat HTTP metodit. OPTIONS metodia voidaan käyttää selvittämään palvelun toiminnallisuutta. CONNECT metodilla vastaanottajaa pyydetään muodostamaan tunneli pyynnössä määritettyyn palvelimeen, ja tunneloinnin onnistuessa toimimaan datapakettien sokeana eteenpäin lähettäjänä asiakkaan ja tunnelin päässä olevan palvelimen välillä. PATCH metodi PUT metodin tavoin päivittää olemassa olevaa resurssin instanssia, mutta sen sijaan että se korvaa koko instanssin rungossa olevalla informaatiolla, se päivittää vanhasta instanssista vain rungossa määritellyt kentät.[15] REST palvelun kannalta oleellisimpia metodeja ovat GET, POST, PUT ja DELETE. Näiden neljän metodin avulla käyttäjä pystyy hakemaan, luomaan, muokkaamaan ja poistamaan resursseja.[14]

Metodit HEAD, GET, OPTIONS ja TRACE ovat niin sanotusti *turvallisia* metodeja. Turvalliset metodit eivät muokkaa palvelimen tilaa millään tavalla, jolloin esimerkiksi GET pyyntöjä voi lähettää huoletta palvelimelle ilman, että käyttäjän tarvitsee pohtia

pyynnön seurauksia. Tämän lisäksi metodit voivat olla *idempotentteja*. Idempotentin pyynnön voi lähettää palvelimelle useita kertoja, mutta ensimmäisen pyynnön jälkeen palvelimen tila ei enää muutu. REST:lle oleellisista metodeista vain POST ei ole idempotentti. Jos POST pyynnön tekee useasti, pyynnössä oleva uusi instanssi luodaan resurssille niin monta kertaa, kuin POST pyyntö menee läpi palvelimelle. Vastakohtana PUT metodi taas muokkaa kohteenaan olevaa resurssia kerran, ja seuraavalla pyynnöllä se muokkaa saman artikkelin samanlaisella tavalla, jolloin mikään ei muutu. Tällä voi olla merkitystä, jos käyttäjä ei huomaa ensimmäisen pyynnön läpimenoa ja toistaa pyynnön. Jos palvelu suorittaa esimerkiksi rahatransaktioita, usean transaktion luonti POST-metodilla voi aiheuttaa toivomattomia sivuvaikutuksia.[14]

Vaikka HTTP:n spesifikaatio määrittelee metodien toimintaperiaatteet ja ominaisuudet tarkasti, jää kuitenkin palvelun kehittäjän vastuulle toteuttaa metodit niiden määritelmän mukaisesti. On siis täysin mahdollista, että GET metodilla palvelu luo uuden instanssin jollekin resurssille, jos palvelu on toteutettu niin. Spesifikaation seuraaminen on kuitenkin järkevää, jotta selain ja palvelun käyttäjä tietävät mitä palvelulta odottaa.

HTTPS on HTTP:n salattu versio. HTTPS käyttää kommunikointiin HTTP:tä, mutta kommunikaatio on salattu käyttäen Transport Layer Securityä (TLS), tai sen edeltäjää, Secure Sockets Layeriä (SSL). HTTPS:n mielekkyys on ilmeinen, kun palvelin käsittelee esimerkiksi luottokorttietoja verkkokauppamaksujen suorittamiseen, mutta myös käyttäjätietojen salaaminen on oleellinen motivaatio HTTPS:n käyttämiseen. Pelkkää HTTP:tä käytettäessä kolmas osapuoli voi esittää palvelinta, jolle informaatiota lähetetään, mutta myös seurata oikealle palvelimelle kulkevaa informaatiota. HTTPS takaa kommunikaation kulkevan oikealle palvelimelle ja tekee liikenteen salakuuntelemisesta mahdotonta. Jotta selaimet voivat luottaa palveluiden salauksen olevan hyvin toteutettu, pitää salauksen sertifikaatti pyytää luotetulta sertifikaatin tarjoajalta. Jos salaus on itse toteutettu, selain kertoo sivun olevan riskialtis hyökkäyksille, vaikka salaus olisikin teoriassa oikein tehty.

Asiakas-palvelin malliin liittyy oleellisesti vielä HTTP:n osalta menettelytapa, jota kutsutaan *saman alkuperän käytännöksi*. Kun selain tekee pyyntöjä palvelimille, se niputtaa tietylle isännälle tehdyt pyynnöt saman *alkuperän* alle. Sopimuksen mukaisesti kahdella URI:lla on sama alkuperä, jos niiden skeema, isäntä ja portti ovat identtiset. Esimerkiksi URI:t http://esimerkki.com ja http://esimerkki.com/kaverit ovat samaa alkuperää, mutta eri skeeman https://esimerkki.com ei ole. Myös aliverkkotunnus, kuten http://admin.esimerkki.com on eri alkuperän URI. Koska asiakasohjelmat tekevät paljon pyyntöjä eri palvelimille, on mahdollista, että tässä tapahtuu asiakkaan ja palvelimen kannalta haitallisia pyyntöjä, joista asiakas ja palvelin eivät ole tietoisia. Saman alkuperän käytäntö määrittelee erilaisia toimintatapoja tilanteille, joissa asiakasohjelma hakee resursseja URI:sta, jonka alkuperä eroaa alkuperäisen palvelimelle tehdyn HTTP-pyynnön alkuperästä. Oleellisin käytäntö työn kannalta on se, että kun jostakin URI:sta haetaan ohjelmakoodia, jota asiakasohjelma suorittaa, tällä ohjelmakoodilla on oikeus

käyttää resursseja vain saman alkuperän URI:sta. Toisin sanottuna ohjelmakoodi ei saa tehdä HTTP-pyyntöjä toisen alkuperän URI:in. Saman alkuperän käytäntöä on kuitenkin mahdollista löysentää palvelimen puolesta, jos palvelin luottaa johonkin palvelimesta itsestään poikkeavan alkuperän lähteeseen. Tätä toimintamallia kutsutaan Cross-Origin Resource Sharing:ksi (CORS). Palvelin määrittelee luotetun lähteen, ja saadessaan pyynnön tästä lähteestä, asettaa pyynnön vastaukseen ylätunnisteen nimeltä Access-Control-Allow-Origin. Näin eri alkuperän lähteet tietävät voivansa luottaa toisiinsa. Esimerkiksi julkisten rajapintojen toimintamalli perustuu kokonaisuudessaan siihen ajatukseen, että pyyntöjä tulee lukemattomista eri alkuperistä. Tässä tilanteessa palvelin asettaa edellä mainitun ylätunnisteen automaattisesti jokaiseen pyyntöön.[16]

2.3.3 Representational State Transfer

REST arkkitehtuurimalli näkyy sitä käyttävälle asiakkaalle URL:ä käyttävänä HTTP rajapintana, mutta se tarjoaa myös sen kehittäjälle joukon positiivisia ominaisuuksia. REST on suorituskykyinen, skaalautuva, yksinkertainen, muunneltava, läpinäkyvä, siirrettävä ja luotettava. REST järjestelmä saavuttaa nämä ominaisuudet seuraamalla REST arkkitehtuurin määrittelevää kuutta rajoitetta.[17]

Ensimmäinen rajoite on asiakas-palvelin mallin tuoma rajoite. Haasteiden eriyttäminen (SoC, engl. Separation of Concerns) tarkoittaa sitä, että järjestelmässä eriytetään toisistaan erilaisia asioita tekevät moduulit.[18] Tässä tapauksessa toisistaan eriytettävät haasteet ovat käyttöliittymän haaste, mikä jätetään asiakkaan tehtäväksi, ja informaation varastoinnin haaste, mikä taas on palvelimen tehtävänä. Tämä seurauksena käyttöliittymän siirrettävyys paranee ja järjestelmän skaalautuvuus kasvaa yksinkertaisemman palvelin komponentin vuoksi. Yleisesti ottaen tämä eriyttäminen mahdollistaa eri komponenttien toisistaan irrallisen kehittämisen.[17]

Toiseksi, asiakas-palvelin kommunikaatiota rajoitetaan tilattomuudella, eli asiakkaan istunnosta ei ylläpidetä tietoa palvelimella. Jokainen asiakkaan pyyntö palvelimelle käsittää itsessään tiedon asiakkaan tilasta, joka säilyy vain asiakkaan päässä. Palvelin voi kuitenkin siirtää informaatiota tietokantaan hetkelliseen tiedon ylläpitämiseen autentikointi tarkoituksessa. Asiakas liikkuu tilasta toiseen tekemällä palvelimelle pyyntöjä ja voi olla siirtymätilassa pyynnön ollessa vielä aktiivisessa käsittelyssä. Asiakkaan tila sisältää linkkejä, joita voidaan käyttää uuteen tilaan siirtymisessä.[17]

Asiakas-palvelin järjestelmä voi olla kerrostettu, eli asiakkaan ja palvelimen välillä voi olla välittäjäpalvelimia auttamassa varsinaista HTTP-pyynnön kohteena olevaa palvelinta ilman, että asiakas huomaa tätä. Näin palvelun skaalautuvuutta voidaan lisätä mahdollistamalla kuorman tasapainotusta ohjaamalla pyyntöjä vähemmän käytetyille palvelinresursseille ja hyödyntää jaettua välimuistia. Välittäjäpalvelin voi myös toimia turvallisuus käytäntöjen ylläpitäjänä.[17]

Sekä asiakas, että asiakkaan ja palvelimen välissä olevat välittäjäpalvelimet voivat tallentaa palvelimen vastauksia välimuistiin. Tämän vuoksi neljäs rajoite määrittää, että vastauksien pitää kertoa asiakkaalle ovatko ne tallennettavissa välimuistiin vai ei. Tämän seurauksena asiakas ei käytä vanhentunutta tai sopimatonta informaatiota myöhempien pyyntöjen yhteydessä. Hyvin ylläpidetty välimuistin hallinta poistaa osittain tai kokonaan tarpeen asiakkaan ja palvelimen väliseen kommunikaatioon, mikä parantaa skaalautuvuutta ja suorituskykyä.[17]

Palvelin voi tarjota asiakkaalle toiminnallisuutta suoritettavan koodin muodossa. Esimerkkinä tästä on asiakasohjelmalla suoritettava JavaScript koodi. Tämä ei ole suoranainen rajoite, vaan REST:n kannalta vapaaehtoinen lisäominaisuus.[17]

Kuudes ja kaikkein perusteellisin rajoite on yhtenäisen rajapinnan rajoite. Yhtenäinen rajapinta yksinkertaistaa ja erottaa arkkitehtuuria, mikä myös on komponenttien erillisen kehittämisen mahdollistaja. Yhtenäinen rajapinta koostuu neljästä osasta:

1. Yksittäiset resurssit ovat identifioitavissa pyynnöstä, verkkopalveluiden tapauksessa URL:sta. Resurssit itsessään ovat käsitteellisesti eri asia kuin palvelimelle palautettu representaatio kyseisestä resurssista. Palvelin lähettää dataa tietokannasta esimerkiksi HTML:nä, XML:nä tai JavaScript Object Notationina, mutta tämä data ei suoraan vastaa tietokannan sisältämää informaatiota.
2. Resurssit ovat muokattavissa niiden representaatioiden kautta. Asiakasohjelman hallussa on tarpeeksi tietoa, tai metatietoa resurssista, jotta sen muokkaaminen tai poistaminen on mahdollista.
3. Jokainen viesti sisältää tarpeeksi informaatiota kuvaamaan, miten viesti tulee käsitellä. Esimerkiksi viestin sisältämän Internet media tyypin perusteella voidaan päätellä, miten viestin informaatiota luetaan.
4. Asiakasohjelma tarvitsee vain alustavan URL:n, jonka jälkeen asiakasohjelman pitää pystyä palautetusta hypermediasta löytämään käytettävissä olevat toiminnat ja resurssit. Uusiin toimintoihin ja resursseihin voidaan siirtyä vastauksesta löytyvien hyperlinkkien avulla. Asiakasohjelman ei siis tarvitse etukäteen tietää mitään palvelimen ja sen sisältävän informaation rakenteesta.[17]

2.4 Langattomat verkot

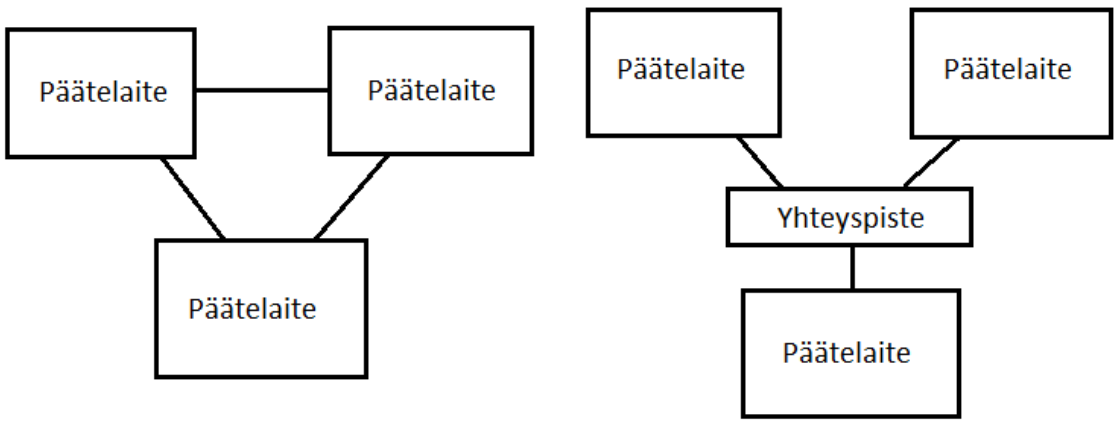
Tässä kappaleessa esittelemme langattomien lähiverkkojen ja langattomien laajaverkkojen toimintaa niiltä osin kuin ne vaikuttavat sovelluksen toimintaan ravintolaoloissa.

2.4.1 WLAN

Wireless Local Area Network (WLAN) on *langaton lähiverkko*, joka yhdistää kahden tai useamman laitteen toisiinsa käyttäen langattomia jakelumenetelmiä rajatun alueen, kuten kodin tai toimiston sisällä. Modernit WLAN:t pohjautuvat Institute of Electrical and Electronics Engineersin määrittelemään 802.11-standardiin langattomien lähiverkkojen toteuttamisesta, mutta WLAN:n rinnalle on kehitetty muitakin standardeja, jotka eivät ole saaneet samanlaista suosiota. Wi-Fi on WLAN toteutuksien brändi-nimi, jota käytetään kaupallisiin tarkoituksiin.

WLAN-päätelaitteet voivat keskenään muodostaa Ad-hoc-verkon, jossa päätelaitteet keskustelevat suoraan toistensa kanssa, kuten kuvan 2.3 vasemmanpuoleisessa topologiassa nähdään. Jos WLAN-työasema tarvitsee pääsyn langalliseen verkkoon, käytetään infrastruktuuriverkkoa. Koska tämä tarve täyttyy työssä, keskitymme vain infrastruktuuriverkkoon.[19]

Infrastruktuuriverkossa päätelaitteet liittyvät kuvan 2.3 oikeanpuoleisen topologian mukaisesti *yhteyspisteeseen*, joka välittää informaatiota langattoman WLAN-verkon ja langallisen verkon välillä. Langallinen verkko voi olla yksityinen rajattu lähiverkko tai osa esimerkiksi Internetiä. Yhteyspiste tekee itsensä näkyväksi sen peittoalueilla oleville päätelaitteille lähettämällä majakkasanoman, jossa kulkee mukana myös yhteyspisteen parametrit. Vastaavasti päätelaitteen WLAN-verkkosovittimen ollessa aktiivisena, se etsii verkkoa kuuntelemalla mahdollisia majakkasanomia. Päätelaitteelle tärkein parametri on verkon Service Set Identifier (SSID), eli *langattoman lähiverkon verkkotunnus*. SSID-tunnus on jokaiselle verkolle ainutlaatuinen arvo, jonka avulla päätelaite tunnistaa haluamansa verkon mahdollisesti useamman päällekkäisen verkon joukosta. Päätelaite valitsee haluamansa verkon ja asettaa parametrinsa vastaamaan yhteyspistettä, jonka jälkeen yhteyspiste tunnistaa päätelaitteen ja rekisteröi sen liityntätunnuksen ja osoitteen verkossa.[19]



Kuva 2.3 *Ad-hoc- ja infrastruktuuriverkko*

WLAN-verkon kykyä toimia voidaan mitata useammalla tavalla. Verkon siirtonopeus on ilmeisin, mutta verkon viive ja kehysten häviäminen matkalla ovat myös oleellisia parametreja verkon toimivuuden kannalta. Tosiaikaisten järjestelmien tilanteessa viive ja kehysten häviäminen ovat erityisen tärkeässä roolissa, sillä viiveen kasvaessa ja kehyksien hävitessä signaalin perille tulemisessa kestää entistä kauemmin. WLAN-verkon nopeus on käytännössä usein vain 60-70% teoreettisestä maksimistaan, sillä käytännön rajoitteet hidastavat verkon toimintaa. Näitä rajoitteita ovat muun muassa radioaaltojen heikentyminen, radiohäiriöt, 802.11-standardin määrittelemän protokollan tasolla tapahtuvat kehysten uudelleenlähetykset ja muut lähellä olevat WLAN-laitteet.[19]

2.4.2 3G

Third Generation (3G), eli kolmannen sukupolven matkapuhelinverkko, on *langaton laajaverkko*. Toisin kuin langaton lähiverkko, laajaverkko on tarkoitettu kattamaan pidempiä välimatkoja, kuten kokonaisia maita tai maanosia. Usein langaton laajaverkko toimii langattoman lähiverkon korvaajana, kun käyttäjä siirtyy langattoman lähiverkon kantavuuden ulkopuolelle, esimerkiksi ulkotiloihin. Langattomat laajaverkot kattavat myös sisätilat, mutta usein niiden kuuluvuus on heikompaa sisätiloissa. Langattoman laajaverkon etu on sen suuri katvealue ja matalammat käyttökustannukset, vaikka sen suorituskyky voikin rajoittain olla huonompi. Huonompikin suorituskyky on kuitenkin parempi kuin se, että ei olisi yhteyttä ollenkaan.[20]

3G-verkko kykenee vähintäänkin 2 megatavua/sekunti tiedonsiirtonopeuteen. WLAN-verkolla taas riippuen 802.11-standardin versiosta voi olla huomattavasti korkeammat tiedonsiirtonopeudet. 3G hyödyntää kuitenkin olemassa olevia linkkimastoja ja jakelujärjestelmiä, mikä tarkoittaa 3G-verkon infrastruktuurin olevan jo olemassa, mikäli WLAN-yhteyspisteitä ei ole saatavilla.[20]

3. MÄÄRITTELY

Tässä kappaleessa käydään läpi ravintoloiden verkkosovelluksen käytössä kohtaamia ongelmia ja esitetään hybridisovellus ratkaisuna kohdattuihin ongelmiin. Lisäksi kappaleessa esitetään Culinarin tekemiä päätöksiä käytetyistä teknologioista ja toteutusta koskevista reunaehdoista. Suuri osa Culinarin saamista vaatimuksista sovellukselle ei ole tullut suoraan eksplisiittisesti asiakkaalta, vaan ne on ilmennyt ongelmina alkuperäisessä tuotteessa, joita on lähdetty ratkomaan uuden sovelluksen kautta.

3.1 Ravintolassa kohdatut ongelmat

Culinarin tuotteen alkuperäinen toteutus oli tehty verkkosovellukseksi. Tuotteen käyttäminen on siis vaatinut laitteen, jolla saa avattua internet-selaimen, sekä toimivan internet-yhteyden. Käytännössä monessa ravintolassa tämä tarkoitti kannettavaa tietokonetta, jota aiemmin oli käytetty ravintolassa esimerkiksi musiikin soittamiseen tai pöytävarauspalveluiden kautta tulleiden pöytävarauksien hallintaan. Kannettavan tietokoneen käyttäminen kuitenkin osoittautui ongelmalliseksi alusta asti. Ravintolassa tiskin puolella pöytätilaa on rajoitettu määrä, kuten kuvasta 3.1 huomaa. Iso ja kömpelö kannettava tietokone tekevät tästä tilasta vieläkin tukalamman. Tietokone olikin sijoitettu ravintoloissa kauemmaksi tiskiltä sivupöydille. Verkkosovelluksen käyttäminen ja

Kuva 3.1 *Culinarin asiakasravintolan tiskitila*

musiikin toistaminen samalla laitteella ei myöskään toiminut ongelmitta. Kun Culinarin palvelu vastaanottaa tilauksen, se ilmoittaa uudesta tilauksesta toistuvalla äänimerkillä. Koska tietokone oli musiikin toistoa varten kytketty ravintolan äänentoistolaitteisiin,

alkoi uuden tilauksen merkkiääni soida äänentoistolaitteiden kautta. Ravintolan reaktio oli laittaa Culinarin palvelu äänettömälle. Tämä yhdistettynä tietokoneen sijaintiin sivupöydällä tarkoitti sitä, että ravintola ei reagoinut uusiin tilauksiin, koska ne jäivät helposti huomiotta.

Isolla osalla ravintoloista ei kuitenkaan ollut käytössä minkäänlaista laitetta, jolla verkkosovellusta olisi voitu käyttää. Culinarin siis piti tarjota näille ravintoloille verkkosovelluksen lisäksi laite, jolla verkkosovellusta voisi käyttää. Koska tilankäyttö oltiin jo todettu ongelmaksi ja laite ei voisi olla liian suuri rasite ravintoloiden budjetille, oli tabletti helppo ratkaisu. Ravintoloilla oli myös mahdollisuus ostaa oma tabletti, jos he eivät halunneet liisata sellaista Culinarilta. Tabletti huomattavasti pienempänä laitteena voitiin helposti sijoittaa tukitelineen avulla tiskille niin, ettei se haitannut tarjoilijan työskentelyä, kuten kuvasta 3.2 voidaan nähdä. Kuva 3.2 on otettu samalta tiskiltä kuin kuva 3.1.

Kuva 3.2 *Culinarin tabletti tiskillä*

Tabletti osoittautui kohtalaisen hyvin toimivaksi ratkaisuksi, mutta nyt itse verkkosovelluksen heikkoudet alkoivat tulla esille entistä selkeämmin. Palveluun päästäkseen tarjoilijan tuli avata internet-selain, syöttää Culinarin palvelun verkko-osoite osoiteriville ja kirjautua sisään. Ravintolan kiireellisessä ympäristössä tämä monen välivaiheen vaatimus osoittautui huonoksi käytettävyydeksi. Lisäksi selain käyttöympäristönä osoittautui kehnoksi valinnaksi. Tarjoilijan oli liian helppo sulkea selain, avata uusi välilehti tai mennä tabletin aloitusnäytölle, jolloin Culinarin verkkosovellus ei ollut aktiivisesti näkyvillä. Kaikki nämä selaimen käytettävyyden ongelmat johtivat taas kerran siihen, että uusia tilauksia ei ravintolassa huomattu. Selaimen tuottamaa käytettävyysongelmaa yritettiin ratkaista väliaikaisesti käyttämällä kolmannen osapuolen tarjoamaa Kiosk Browser-sovellusta, jolla selaimen sai lukittua

samalle sivulle niin, että käyttäjät eivät voineet vahingossa sulkea selainta tai asettaa sitä taustalle. Tästä seurasi vain uusi ongelma, sillä virhetilanteiden esiintyessä lukitulle palvelulle oli hankala tehdä toimenpiteitä, joilla virhetilanteista voisi palautua normaaliin tilaan. Virhetilanteet olivat kuitenkin tässä vaiheessa palvelun kehittämistä varsin yleisiä ja lukittu selain päätyi usein virhetilanteisiin, joista ravintolassa ei selvitty ilman Culinarin tukea.

Yksinkertaistettuna kaikki nämä ongelmat johtivat siis siihen, että ravintoloiden asiakkaiden tekemät tilaukset eivät tulleet perille ravintolaan tarpeeksi suurella toimintavarmuudella. Koska tilausten digitalisointi on Culinarin palvelun ydin, tähän ongelmaan vaadittiin nopeaa ratkaisua.

3.2 Ratkaisu ravintoloiden ongelmaan

Teknisestä näkökulmasta natiivi sovellus tarjoaa ratkaisun kaikkiin verkkosovelluksen ongelmiin. Android-pohjainen sovellus olisi erityisen hyvä ratkaisu, sillä Culinar oli jo aloittanut Android-tablettien liisaamisen ravintoloille, joilla ei ollut päätelaitetta verkkosovelluksen käyttämiseen. Ravintolan tarvitsee vain ladata natiivi sovellus Androidin sovelluskaupasta, Google Play Storesta. Tämän jälkeen sovellus on aina helposti saatavilla pikakuvakkeena tabletin aloitusnäytöltä. Ravintolan ei enää tarvitse ensin käyttää selainta päästäkseen sisälle Culinarin palveluun. Culinar pystyy myös esiasentamaan sovelluksen uusille ravintoloille, joille liisattua tablettia ei ole vielä keretty lähettää. Näin ravintolan työntekijät voivat tabletin saadessaan suoraan käynnistää sovelluksen ja kirjautua sisään, vähentäen entisestään turhien välivaiheiden määrää. Ravintolan on toki mahdollista sulkea sovellus, tai päätyä aloitusruudulle painamalla androidin koti-painiketta, mutta natiivi sovellus tarjoaa työkalut tämän ongelman ratkaisemiseen.

Verkkosovelluksessa ruokatilaukset on lähetetty suoraan Culinarin palvelinohjelmalta selaimella ajettavalle asiakasohjelmalle. Tässä vaiheessa on käyttäjän vastuulla, että selain on auki ja Culinarin palvelu aktiivinen. Jos selain ei ole aktiivisena tai päällä, ohjelmallisesti ei tälle asialle ole mitään tehtävissä ja tilaukset jäävät piiloon tai saamatta. Natiivi sovellus ratkaisee ongelman push-ilmoitusten avulla. Ilmoitus lähtee palvelinohjelmalta Google Cloud Messaging palveluun, missä ilmoitus odottaa niin kauan, että se tavoittaa päätelaitteen, jolle ilmoitus on tarkoitus lähettää. GCM:ltä tulevat ilmoitukset ovat osa Android-laitteen sisäistä toimintaa ja tämän vuoksi, kun päätelaite vastaanottaa ilmoituksen, se tietää mille sovellukselle ilmoitus kuuluu. Nyt ilmoitus voidaan näyttää Androidin ilmoituskeskuksessa ja ilmoitusta napauttamalla sovellus voidaan automaattisesti herättää aktiiviseksi sovellukseksi tai käynnistää uudelleen, mikäli sovellus on suljettu. Tilauksen läpimenovarmuus kasvaa, koska ravintolan työntekijällä on vähemmän mahdollisuuksia jättää päätelaitetta tilaan, jossa tilaukset jäävät huomiotta.

Ilmoitukset ovat suurin toimintavarmuuden parantaja, mutta ei kuitenkaan ainut natiivin sovelluksen tuoma etu. Natiivi sovellus pääsee käsiksi päätelaitteen näytön asetuksiin ja akun tilan tietoihin. Tavanomaisessa tilanteessa laitteen asetuksissa määritetyn ajan kuluttua näyttö sammuu ja päätelaite menee unitilaan. Unitilassa laite voi akunsäästämisen vuoksi sammuttaa WLAN:n, jolloin tilaukset huomataan vasta laitteen aktivoituessa uudelleen. Natiivi sovellus kuitenkin mahdollistaa näytön jatkuvan päällä pitämisen. Tämä syö kuitenkin paljon akun virtaa, jonka vuoksi samalla on tunnistettava, että laite on lataustilassa kytkettynä latausjohtoon. Jos laite irrotetaan latauskaapelista, näytön sammuminen voidaan taas kytkeä päälle. Lisäksi natiivi sovellus mahdollistaa laitteiden paremman monitoroinnin. Sovellus voi lähettää tietoa tilastaan palvelinohjelmalle ja näin kehittäjillä on parempaa tietoa saatavilla heti mahdollisissa ongelmatilanteissa. Tätä tietoa voidaan myös käyttää ongelmatilanteiden ehkäisemisessä ennen kuin ongelmatilanteet pääsevät muodostumaan. Laitteelta voidaan lähettää akun tila, WLAN/3G signaalin voimakkuus, WLAN-verkon SSID, käytetyn laitteen nimi, käyttöjärjestelmän versio, sovelluksen versio ja tietoa sovelluksessa tapahtuvista asioista, kuten laitteella olevien tilausten tila, tai onko laitteessa äänet päällä.

Teknisten ominaisuuksien puolesta natiivi sovellus on täydellinen ratkaisu Culinarin ongelmiin. Natiivin sovelluksen kehittäminen Androidin kehityskielellä, Javalla, vaatisi kuitenkin täysin uuden projektin aloittamisen ja koko sovelluksen kirjoittamisen tyhjästä. Lisäksi on huomioitava, että Culinarin työntekijöillä ei ole entuudestaan kokemuksesta sovellusten kehittämisestä Androidille, jonka vuoksi myös uusien tietotaitojen opettelemiseen kuluisi paljon aikaa. Aika- ja resurssipulan vuoksi jäljelle jäi vain yksi realistinen vaihtoehto, hybridisovellus. Verkkosovelluksen siirtäminen hybridisovellusmuotoon on huomattavasti nopeampaa, kuin kokonaan uuden sovelluksen kirjoittaminen, sillä hybridisovellus hyödyntää samoja verkkoteknologioita kuin verkkosovellus. Näin kehittäjän on mahdollista keskittyä suoraan tarvittavien uusien ominaisuuksien kehittämiseen, eikä aikaa kulu jo olemassa olevien ominaisuuksien monistamiseen.

Hybridisovelluksen kehittämistä suunniteltaessa huomattiin myös palvelinohjelman rajapinnan kaipaavan uudistamista. Alkuperäistä rajapintaa ei oltu koskaan erityisesti suunniteltu, vaan se kehittyi itsestään tarpeiden mukaan. Hybridisovellus kuitenkin pitää liittää tähän samaan rajapintaan, jonka vuoksi alkuperäistä rajapintaa tulee siistiä ja selkeyttää niin, että vanha verkkosovellus, ja uusi hybridisovellus voivat hyödyntää sitä jatkossa yhtäaikaisesti.

3.3 Käytetyt teknologiat

Tässä kappaleessa käydään läpi työssä käytettyjä teknologioita ja päätöksiä, joidenka vuoksi näihin teknologioihin on päädytty muiden vaihtoehtojen sijaan. Yleisenä valintaperusteena Culinarilla on käytetty kehittämisen helppoutta ja mahdollisimman nopeaa kykyä saada tuote kuluttajien käyttöön. Verkkosovellusten, ja näin myös

hybridisovellusten kehittämiselle on ominaista, että kehitystyössä käytetään hyväksi suurta määrää eri ohjelmistokirjastoja. Koska tässäkin projektissa on päädytty käyttämään paljon erilaisia kirjastoja, esille tuodaan nyt vain oleellisimmat ohjelmistokehykset, joiden varaan koko projekti on rakennettu.

3.3.1 Node.js

Node.js on avoimen lähdekoodin järjestelmästä riippumaton ajonaikainen JavaScript ympäristö. Node.js toteuttaa tapahtumapohjaista arkkitehtuuria, joka kykenee asynkronisiin sisääntulo/ulostulo operaatioihin. Tämä arkkitehtuuri pyrkii optimoimaan näiden operaatioiden läpisyötön ja skaalautuvuuden. Node.js on siis erinomainen valinta verkkosovelluksien HTTP-palvelimena, sillä HTTP-palvelimet voivat joutua yhtäaikaisesti valtavan sisääntulo-operaatioiden määrän kohteeksi HTTP-pyyntöjen muodossa. Tätä varten node.js:n on sisäänrakennettu valmiiksi HTTP-moduuli. Käytännössä kaupallisen käytön node.js sovellukset toimivatkin HTTP-palvelimina. Node.js:stä tekee hyvän valinnan HTTP-palvelimeksi tehokkaan suorituskyvyn lisäksi myös sen ohjelmointikieli, JavaScript. Kaikissa selaimissa on sisäänrakennettuna tuki JavaScriptille, jonka vuoksi asiakasohjelmien toiminnallisuus verkkosivuilla ohjelmoidaan aina JavaScriptillä. Ennen node.js:ää verkkosivujen kehittämiseen on vaadittu kahden eri ohjelmointikielen osaaminen, sillä JavaScriptillä toimivia palvelin-ympäristöjä ei ollut olemassa. Node.js kuitenkin mahdollistaa verkkosivun kehittämisen pelkällä JavaScriptillä palvelinpuolesta asiakasohjelmaan. Kehittäjän on siis paljon helpompi yhtäaikaisesti kehittää molempia puolia ilman ohjelmointikielen vaihtumista.

Culinarin kahdella kehittäjällä toisella oli ennestään kokemusta node.js:stä ja toisella djangosta, jonka vuoksi nopean liikkeelle pääsemisen saavuttamiseksi HTTP-palvelimen tuli olla jompikumpi näistä kahdesta. JavaScript node.js:n ohjelmointikielenä täytti kuitenkin helppouden vaatimuksen suhteessa muihin ympäristöihin, joista yksikään ei käytä JavaScriptiä kielenään. Djangon ohjelmointikielenä toimii python. Tämän lisäksi node.js:n suosion vuoksi sille on rakennettu suuri määrä erilaisia paketteja, jotka toteuttavat laaja-alaisesti ratkaisuja kehittäjien tarpeisiin. Node.js:n pakettienhallinta työkalu npm käsittää 350 000 paketin tuen työn kirjoittamisen ajan hetkellä. Näistä syistä node.js valittiin Culinarin sovellusten HTTP-palvelimeksi.

Node.js itsessään tarjoaa vain suhteellisen matalan tason työkalut ohjelmistojen rakentamiseen. Tästä johtuen node.js:lle on kehitetty useita eri ohjelmistokehyksiä, joiden tarkoituksena on helpottaa node.js sovelluksien kehitystyötä. Nämä ohjelmistokehykset vaihtelevat kevyistä ja yksinkertaisista ratkaisuista monimutkaisiin, jokaisen kehitystyön vaiheen kattaviin ratkaisuihin. Mitä isompi ja raskaampi ohjelmistokehys on kyseessä, sitä enemmän se tarjoaa liikkeelle pääsemiseksi valmista ohjelmakoodia, jonka päälle voi alkaa kehittää omaa ohjelmaa. Ajattelematta asiaa enempää, tämä kuulostaa hyvältä asialta. Varsinkin, kun tämä tuntuisi täyttävän kehittämisen helppouden ja nopean tuotteen julkaisemisen vaatimukset. Aikaa kuluu kuitenkin ohjelmistokehyksen omien

erikoisuuksien opettelemiseen ja valmis ohjelmakoodi saa helposti aikaan tilanteen, jossa kehittäjä ei ole täysin varma miksi asiat toimivat niin kuin ne toimivat. Tämä taas johtaa siihen, että aikaa kuluu valmiin ohjelmakoodin tutkimiseen. Tästä johtuen Culinarilla päädyttiin Express ohjelmistokehykseen, joka on erittäin minimalistinen ja suorituskyvyltään tehokas. Vaikka alkuun pääsemiseksi joutuu näkemään hieman enemmän vaivaa kuin raskaammilla ohjelmistokehyksillä, aikaa ei tämän alun jälkeen kulu turhaan ihmettelyyn ja näin säästetään ohjelmistokehittäjän aikaa. Express on myös yksi node.js:n suosituimpia ohjelmistokehyksiä, minkä vuoksi sille löytyy erittäin hyvä tuki ja paljon materiaalia auttamaan ongelmatilanteissa.

Node.js ja Express toimivat jo Culinarin verkkosovelluksen HTTP-palvelimena, mutta nyt ne tulisivat myös toimimaan hybridisovelluksen palvelimena ja tämän seurauksena HTTP-palvelimen rajapinta piti suunnitella ja mukauttaa sellaiseksi, että se toimii hyvin useamman sovelluksen käytössä.

3.3.2 AngularJS

Selaimessa ajettava asiakasohjelma kehitetään HTML:n, CSS:n ja JavaScriptin yhdistelmänä. Näitä standardeja ei alun perin ole kuitenkaan kehitetty nykyisen kaltaisten dynaamisten verkkosovellusten kehittämiseen. HTML esimerkiksi kehitettiin tieteellisten dokumenttien jakamiseen. Dynaamisten verkkosivujen kehittäminen on kuitenkin näillä työkaluilla teoriassa mahdollista ja esimerkiksi JavaScriptin avulla HTML sivuja voi muokata monipuolisesti. Käytännössä kehittäjän näkökulmasta tämä on todella raskasta ja paljon aikaa vaativaa työtä, jos kehitettävän sovelluksen on tarkoitus tehdä muutakin, kuin renderöidä informaatiota. Näitä puutteita paikkaamaan on myös asiakasohjelmalle kehitetty valtava määrä erilaisia ohjelmistokehyksiä, joiden tarkoituksena on tehdä esimerkiksi HTML:n muokkaamisesta helpompaa ja ymmärrettävämpää. Erityisesti nämä ohjelmistokehykset pyrkivät noudattamaan SPA-mallia, jonka vuoksi ne ovat hyvä valinta verkkosovelluksen kehittämiseen.

Culinarin kehittäjillä ei ollut ennestään kokemusta asiakasohjelmien ohjelmistokehyksistä, jonka vuoksi päätös piti tehdä perustuen muihin tekijöihin. AngularJS oli noussut yliopiston kurssilla esille malliesimerkkinä tällaisesta ohjelmistokehyksestä, jonka vuoksi se oli ainut ohjelmistokehys, joka Culinarilla entuudestaan tiedettiin nimeltä. AngularJS on Googlen kehittämä ohjelmistokehys, joka tarjoaa laajan valikoiman työkaluja verkkokehittämisen haasteiden ratkaisemiseen. Googlen palveluna sillä on taustallaan ison organisaation tuki, sekä laaja verkkokehittäjien yhteisö. Viime kädessä ohjelmistokehyksiä on hankala vertailla ilman kokemusta niillä kehittämisestä. Tästä johtuen AngularJS tuntui hyvältä valinnalta, sillä laaja tuki tekee ennalta näkemättömien ongelmien ratkaisemisesta helpompaa.

3.3.3 Ionic

Valinta natiivin sovelluksen ja hybridisovelluksen väliltä ei ollut heti ilmeinen, kun verkkosovellus todettiin riittämättömäksi ratkaisuksi. Natiiveilla sovelluksilla on taustalla pidempi historia ja tuki. Hybridisovellukset taas ovat uusi suuntaus, ja sen taustalla oleva teknologia ei ole kerennyt kypsyä pisteeseen, jossa toimintamallit ja –tavat olisivat vakiintuneet. Hybridisovellus on helppo ottaa käyttöön moneen eri mobiilikäyttöjärjestelmään, mutta Culinarin ensisijainen tarve oli vain saada sovellus Android-käyttöjärjestelmälle, koska Culinar liisasi nimenomaan Android-tabletteja ravintolalle. Kehittämisen helppous ja nopea tuotteen julkaiseminen kuitenkin kallistivat päätöksen hybridisovelluksien puolelle kahdesta syystä. Hybridisovellukset kehitetään käyttäen samoja teknologioita, kuin verkkosovellukset. Koska Culinar oli jo toteuttanut verkkosovelluksen, tulevan hybridisovelluksen kehittäjällä oli jo vahva ymmärrys ja osaaminen siitä, mitä ollaan tekemässä. Toiseksi, kehittäjien suosiossa oleva hybridien kehittämiseen tarkoitettu ohjelmistokehys Ionic sattui käyttämään hyväkseen myös AngularJS:ää. Käytännössä tämä tarkoitti sitä, että verkkosovelluksen ohjelmakoodi oli uudelleenkäytettävissä hybridisovelluksen kehittämiseen. Sen lisäksi, että hybridisovelluksen toteuttaminen oli näillä puitteilla huomattavasti nopeampaa kuin natiivin sovelluksen kehittäminen, uusien ominaisuuksien lisääminen sekä verkkosovellukseen, että hybridisovellukseen pystyttiin toteuttamaan lähes ilman ylimääräistä vaivaa.

Ionic ohjelmistokehys on ilmainen avoimen ohjelmistokoodin projekti. Ionicillä toteutettu hybridisovellus voidaan ottaa käyttöön Androidille, iOS:lle tai Windows Phonelle. Ionicissä on tuki yli 70:lle natiivin laitteen ominaisuudelle, mikä mahdollistaa natiivin sovelluksen kaltaisen kokemuksen toteuttamisen. Vaikka Ionic julkaistiin vasta 2013, on sillä kehitetty Ionicin perustaneen Drifty Co:n mukaan yli 3 miljoonaa sovellusta. Ionic tarjoaa asiakasohjelmien kehittämiseen visuaalisia elementtejä ja rakennuspalikoita, joita hyväksikäyttäen sovelluksista on helppo rakentaa natiivien sovelluksien kaltaisia. Puhelimen natiiveihin ominaisuuksiin Ionic pääsee käsiksi Apache Cordovan avulla. Cordovan tehtävä on toimia tulkkina Ionicin käyttämän verkkoteknologian ja puhelimen natiivin ympäristön välillä. Cordova ei siis varsinaisesti ole osa Ionicia, vaan se on erillinen tuote, jota käyttävät myös monet muut Ionicin kaltaiset hybridisovelluksien ohjelmistokehykset. Cordova antaa Ionicille pääsyn Androidin web view komponenttiin, jonka avulla Ionic pystyy myös rakentamaan varsinaisen sovelluksen näkymän verkkoteknologioilla. (tähän sitaatti ionicin sivulle)

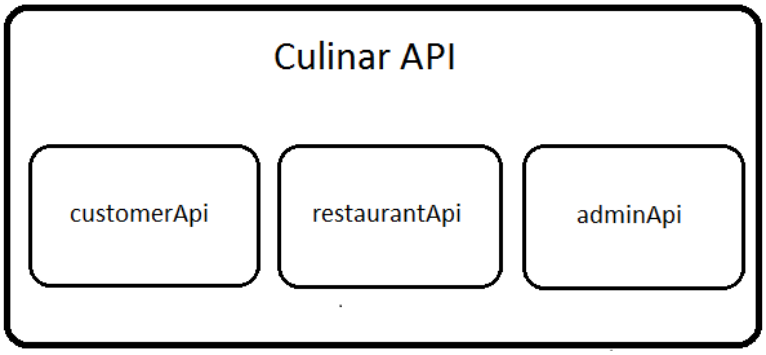
3.4 Palvelinohjelman rajapinta

Culinarin verkkosovelluksen kehityksen alkuvaiheessa ei ollut täysin selvää minkälaisia vaatimuksia sovelluksen tulee täyttää. Koska kyseessä ei ole asiakkaan tilaama ja ennalta määrittelemä tuote, vaan palveluna asiakkaan tarpeita ennakoiden toteutettu sovellus,

näiden tarpeiden löytäminen ja ratkaiseminen on ollut iteratiivinen prosessi. Asiakas on antanut parannusehdotuksia ja palautetta sovelluksesta, ja tähän palautteeseen on reagoitu kehittämällä palvelusta uusia versioita. Kun hybridisovelluksen kehittäminen aloitettiin, tämä prosessi oli kuitenkin edennyt niin pitkälle, että asiakasohjelman ja palvelinohjelman välinen tiedonsiirto oli vakioitunut, eikä enää ollut isojen muutosten kohteena. Teknisestä näkökulmasta tämä rajapinta oli kuitenkin varsin kaoottinen, eikä noudattanut mitään hyvän maun mukaista mallia. Hybridisovelluksen kuitenkin pitäisi pystyä kommunikoimaan saman rajapinnan kanssa, jonka vuoksi rajapinta päätettiin kirjoittaa uusiksi. Selkeys, ymmärrettävyys ja haasteiden eriyttäminen olivat uudelleenkirjoittamisen lähtökohdat.

Selkeys ja ymmärrettävyys tähtäävät käytännössä siihen, että rajapinnan dokumentaatio voitaisiin luovuttaa jollekin ulkopuoliselle henkilölle tai taholle, ja he kykenisivät käyttämään ja tulkitsemaan tätä rajapintaa. Tämä sen vuoksi, että jos jossain vaiheessa Culinar pääsee kasvamaan ja työntekijöitä palkataan lisää, uuden työntekijän pitää kyetä mahdollisimman vähällä vaivalla ymmärtämään kehitettyä ohjelmistoa, jotta muiden työntekijöiden aikaa ei kulu ohjelmiston monimutkaisuuden opettamiseen ja työntekijä pääsee mahdollisimman nopeasti tekemään kehittävää työtä. Toiseksi, jossain vaiheessa voi olla mahdollista, että kolmannen osapuolen kehittäjä haluaa hyödyntää ravintoloihin päin olevaa rajapintaa ja tällainen yhteistyön mahdollisuus on pidettävä Culinarin etenemisen kannalta auki.

Haasteiden eriyttäminen tarkoittaa sitä, että ohjelmiston arkkitehtuuri jaetaan toisistaan riippumattomiksi moduuleiksi, jotka toteuttavat omaa tarkalleen rajattua tehtäväänsä.[18] Rajapinnan suunnittelussa havaittiin kolme selkeää moduulia, jotka päätettiin eriyttää toisistaan: Ravintolan asiakkaille näkyvä rajapinta, ravintolalle näkyvä rajapinta, sekä järjestelmää ylläpitäville kehittäjille näkyvä rajapinta. Jaon loogisuus tulee esille esimerkiksi siinä, että hybridisovelluksella on tarve käyttää vain ravintolalle näkyvää rajapintaa. Kuvassa 3.3 näkyy visualisoituna rajapinnan jako kolmeen osaan ja niiden arkkitehtuurin sisäiset nimet. Nimissä käytetty lyhenne API tulee englanninkielen termistä Application Programming Interface, jota käytetään yleisesti tarkoittamaan verkkopalveluissa rajapintaa.



Kuva 3.3 *Culinarin rajapinnat*

Koska kyseessä on HTTP API, kutsut tehdään rajapintaan HTTP-kutsuina ennalta määrättyihin URL-osoitteisiin. Alkuperäisessä rajapinnassa kutsut olivat muotoa: https://culinar.fi/createOrder. Osoitteen polku oli yksinkertainen, ja kertoi suoraan sen, mikä kyseisen osoitteen funktio on. Ongelma on kuitenkin se, että eri moduuleilla voi olla tarve samankaltaiselle funktiolle, mutta tarvitsee erilaiset käyttäjäoikeudet tai vastaanottaa eri parametrit. Ensimmäinen askel rajapintojen erottamiseen oli se, että osoitteen polun alkuun lisättiin kyseisen rajapinnan nimi, jolloin aikaisemmin kuvatusta osoitteesta muodostui https://culinar.fi/customerApi/createOrder. Nyt eri rajapinnoilla voi olla sama funktio, mutta osoite jakaa funktion rajapinnan nimen perusteella. Yleisen selkeyden vuoksi funktion käsittelemä resurssi ja operaatio eriteltiin vielä polussa, jolloin lopullinen osoite muodostui seuraavasti: https://culinar.fi/customerApi/order/create. Näin osoitteesta muodostuu hierarkia: ensin kerrotaan rajapinta, sitten käsiteltävä resurssi ja lopuksi itse operaatio. Tämä myös kuvastaa kyseisen rajapinnan sisäistä moduulijakoa. Rajapinnan sisällä on tilauksia käsittelevä order moduuli, jonka create funktiota ollaan käyttämässä.

Tyypillisessä REST rajapinnassa funktion toiminta määritellään HTTP-pyynnön tyypin perusteella. Tilauksen luonnin esimerkissä se tarkoittaisi, että polkuun customerApi/order tehtäisiin POST-pyyntö. Culinarilla päätettiin poiketa tästä tavasta, koska funktion nimen lisääminen polkuun dokumentoi itsensä paremmin. Lisäksi tämä mahdollistaa, että kyseiseen resurssiin voidaan tehdä operaatioita, jotka eivät suoraan vastaa HTTP-pyyntöjen luku-, kirjoitus-, päivitys-, tai poisto-operaatioita. Tästä huolimatta pyrkimys oli pitää HTTP-pyyntöjen tyypit operaatioita vastaavina, jolloin kyseinen tilauksen luonti tehdään HTTP POST-pyyntönä.

HTTP-rajapinnan uusiminen ei vaikuta varsinaisten funktioiden toimintaan. Funktioiden parametrit ja käyttäjäoikeusvaatimukset pysyivät samoina. Ainut parametreihin tehty muutos oli se, että parametrien nimiä muokattiin yhtenäisemmäksi eri funktioiden välillä. Jossain tilanteessa esimerkiksi ravintolaa kuvaava uniikki tunnus saattoi kulkea nimellä restaurant, joskus taas nimellä restaurantId. Yhtenäistämisen tuloksena kaikissa tilanteissa päätettiin käyttää pelkkää resurssin nimeä restaurant. Palvelinohjelmaan rajapinnan uusiminen ei vaikuttanut toiminnallisella tasolla, mutta arkkitehtuurillisesti moduulijakoa ei oltu aikaisemmin toteutettu. Nyt jokaiseen rajapintaan kuuluvat polut määriteltiin erillään muista rajapinnoista ja niihin kuuluvat moduulit sijoitettiin tiedostorakenteessa omaksi kokonaisuudekseen. Tämän seurauksena kyseisen rajapinnan kehittäminen helpottui, sillä ne eivät ole enää keskenään sekaisin.

3.5 Hyväksymättömien tilausten protokolla

Ravintolan asiakkaan näkökulmasta on erityisen tärkeää, että Culinarin palvelu onnistuu välittämään viestejä ravintolan ja asiakkaan välillä mahdollisimman tehokkaasti. Jotta asiakaspalvelukokemus olisi hyvä, asiakkaan tulee saada ilmoitus tilauksen läpimenosta niin nopeasti, että asiakkaan ei tarvitse huolehtia onko tilaus varmasti mennyt perille

ravintolaan asti vai ei. Tilauksen hyväksymisen pitkittyessä asiakas voi kokea esimerkiksi puhelimen olevan toimintavarmuudeltaan parempi vaihtoehto, eikä hän seuraavalla kerralla halua turvautua huonommin toimivan verkkotilauspalvelun käyttämiseen. Ravintoloissa on kuitenkin usein kehnot langattomat verkkoyhteydet, eikä ravintolan väki aina kerkeä heti reagoimaan tehtyihin tilauksiin. Tästä johtuen applikaation kehittämisen yhteydessä piti myös suunnitella protokolla, joka kertoo kuinka toimia, kun tilausten hyväksyminen viivästyy.

Protokollan lähtökohdaksi päätettiin ajanhetki, jolloin ravintolan tulee hyväksyä tilaus. Tämä ajankohta on heti, jos ravintola on auki tilauksen luonnin hetkellä. Jos ravintola on kiinni, tilaus pitää hyväksyä ravintolan aukeamisen hetkellä. Tähän ajan hetkeen viitataan kappaleessa *optimaalisena hyväksymisaikana*.

Ensimmäinen protokollaan liittyvä kysymys käsitteli tilauksen ravintolalle toimittamisajankohtaa. On ilmeistä, että ravintolan ollessa auki, tulee tilaus saada välittömästi laitteelle. Etenkin jos asiakas on pyytänyt tilauksen toimitettavaksi niin pian kuin mahdollista, on tilauksen oltava laitteella heti. Tilanne on epäselvempi, kun asiakas tekee tilauksen ravintolan ollessa kiinni. Jos tilaus toimitetaan laitteelle tällöin, on erittäin todennäköistä, ettei ravintolassa kukaan ole vastaanottamassa tilausta. Vaihtoehtoina on joko lähettää tilaus laitteelle ravintolan auetessa, tai lähettää tilaus välittömästi laitteelle, mutta muuttaa tavallista toimintamallia niin, ettei laite soita äänimerkkiä turhaan. Jälkimmäinen vaihtoehto todettiin paremmaksi, koska se ei ole niin riippuvainen ravintolan verkkoyhteyden toimintavarmuudesta. Jos tilaus lähetetään laitteelle vasta ravintolan auetessa ja tällä hetkellä ravintolassa on verkkoyhteys poikki, jää tilaus näkemättä yhteysongelmien ajaksi. Kun tilaus välitetään ravintolan päätelaitteelle heti, riittää että jossakin vaiheessa tilauksen luomisen, ja ravintolan aukeamisen välissä laite on yhteydessä verkkoon.

Seuraava kysymys käsittelee toimenpiteitä, jotka suoritetaan, kun aikaa alkaa kulua optimaalisesta hyväksymisajasta. Alkuperäinen lähtökohta oli, että jokainen tilaus olisi saatava ravintolaan perille riippumatta ongelmista tilauksen läpimenemisessä. Tämä tarkoitti käytännössä sitä, että jos tilausta ei saatu toimitettua laitteelle ohjelmallisesti, Culinarin työntekijä soitti tilauksen ravintolaan, ja kuittasi tilauksen etänä. Tämä oli kuitenkin naiivi toimintamalli, johon sisältyi ajatus siitä, että jossain vaiheessa järjestelmä olisi mahdollista toteuttaa niin hyvin, että tilaukset menisivät aina perille. Ravintoloiden määrän kasvaessa tuli kuitenkin eteen se tosiasia, että kaikkia ravintoloiden verkkoyhteysongelmia ei voi ratkaista, ja että tilausten soittaminen ravintolaan oli liian aikaa vaativaa työtä pienelle kolmen hengen yritykselle. Tästä johtuen uusi protokolla piti toteuttaa niin, että nämä tilanteet voidaan käsitellä ilman oikean ihmisen osallisuutta.

Aikaisemmin ihmisen osallisuutta oltiin pyritty vähentämään automaattisilla puhelinsoitoilla. Kun optimaalisesta hyväksymisajankohdasta oli kulunut jonkin verran aikaa, ravintolaan soitettiin automaattisesti puhelu, jossa kone kertoi englanniksi

ravintolalle, että heillä on hyväksymätön tilaus odottamassa. Tässä lähestymistavassa huomattiin kuitenkin ongelmia. Yleisemmästä näkökulmasta jokainen puhelinsoitto maksaa Culinarille, jonka vuoksi liiketoiminta mielessä se ei ole skaalaava ratkaisu. Lisäksi soittotoiminnallisuuden laajentaminen ulkomaille olisi hankalaa, koska jokaisessa uudessa maassa pitäisi rekisteröidä uusi puhelinnumero jota käyttää sen maan ravintoloille, ja tämä taas tuo lisää työtä ja monimutkaistaa teknologian arkkitehtuuria. Puhelinsoitto osoittautui myös ravintoloiden näkökulmasta ongelmalliseksi. Oletetaan tilanne, jossa ravintolassa on verkkoyhteys poikki eikä tilausta ole saatu tämän vuoksi laitteelle. Nyt ravintolaan tulee vähän ajan kuluttua puhelu, jossa kerrotaan, että heitä odottaa tilaus. Tämä vain sai ravintolan työntekijät turhautumaan, koska he tiesivät, että asiakas on tehnyt tilauksen, mutta he eivät voineet tehdä asialle mitään. Ratkaisun pitäisi siis toimia myös niin, että se ei luo lisää stressiä ravintolalle.

Protokolla päätettiin toteuttaa siltä lähtökohdalta, että tilauksia saa jäädä toteuttamatta, kunhan Culinarin toimintamalli näissä tapauksissa on tarpeeksi hyvä. Käytännössä tämä tarkoittaa tilauksen hylkäämistä automaattisesti sopivana ajankohtana seuraavan mallin mukaisesti.

- Kuten aikaisemmin todettiin, tilaus pyritään lähettämään välittömästi laitteelle. Kun sovellus huomaa saaneensa uuden tilauksen, se lähettää palvelimelle ilmoituksen tilauksen saapumisesta. Jos tämä ilmoitus ei saavu viiden minuutin kuluttua optimaalisesta hyväksymisajasta, tilaus hylätään. Tässä tilanteessa laitteen internetyhteys on ollut pois päältä jo viiden minuutin ajan, eikä se todennäköisesti ole palaamassa, jonka vuoksi on turha odottaa pidempään.
- Jos palvelin on saanut ilmoituksen tilauksen saapumisesta, voidaan olettaa, että ravintolan työntekijöillä on mahdollisuus nähdä uusi tilaus ja odotamme pidempään, sillä ravintolassa voi olla kiireellistä. Kuitenkin kymmenen minuutin kuluttua optimaalisesta hyväksymisajasta palvelin lähettää testipyynnön laitteelle. Jos pyyntöön ei saada vastausta, palvelin hylkää tilauksen. Tämä tarkoittaa, että ravintolan internetyhteys on katkennut tilauksen vastaanottamisen jälkeen.
- Lopulta jos tilausta ei ole hyväksytty 15 minuutin kuluttua optimaalisesta hyväksymisajasta, tilaus hylätään. Laitteen internetyhteys toimii kyllä, koska se on päässyt edellisistä tarkistuksista läpi, mutta ravintolassa on joko todella kiireistä tai jonkun tuntemattoman syyn vuoksi ravintola ei vain hyväksy tilausta. Asiakkaan on turha antaa odottaa loputtomiin tilauksensa kanssa, jos ravintola ei tee asialle mitään.

Automaattisen hylkäämisen lähestyessä ravintolan päässä laitteella näytetään varoitus, jossa kerrotaan, milloin hylkäys tapahtuu. Vaikka ilmoitus voi aiheuttaa närkästystä työntekijöissä, koska he eivät mahda mitään internetyhteyden katkeamiselle, on se silti parempi tilanne kuin se, että tilaus häviää laitteelta yhtäkkiä automaattisen hylkäyksen yhteydessä. Varsinkin, jos ravintolassa ollaan juuri kirjoittamassa tilausta ylös ennen

tilauksen hyväksymistä, olisi tilauksen yhtäkkinen katoaminen erittäin huono asia. Hylkäämisen yhteydessä asiakkaalle lähetetään sekä sähköpostiviesti ja puhelimeen tekstiviesti, joissa kerrotaan, että hänen tilaus on hylätty johtuen ongelmista vastaanottaa tilaus. Mikäli asiakas on maksanut tilauksensa etukäteen kortilla, maksutapahtuma mitätöidään ja rahat palautetaan asiakkaan tilille.

4. IMPLEMENTAATIO

Tässä kappaleessa käydään läpi edellisessä kappaleessa mainittujen vaatimusten ja määrittelyjen pohjalta tehty implementaatio. Vaikka työssä määrittely ja implementaatio on jaettu kahteen osaan selkeyden vuoksi, todellisuudessa implementaatio on toteutettu paljon käsi kädessä määrittelyn kanssa. Käytännössä perinteistä vesiputousmallia, jossa eri vaiheet kuten määrittely ja toteutus tehdään järjestyksessä, on erittäin hankala toteuttaa, kun uusia vaatimuksia selviää otettaessa uusia ominaisuuksia käyttöön. Tällaisissa tilanteissa määrittelyä on jouduttu muokkaamaan ja toteutusta parantamaan uuden määrittelyn pohjalta. Tässä kappaleessa kerrotaan myös näistä iteratiivisista vaiheista, kun se on oleellista työn kannalta.

Hybridisovelluksen lähtökohtana oli toteuttaa vain Androidilla toimiva sovellus, mutta implementoinnin alussa iOS-sovelluksen kehittämistä pidettiin myös todennäköisenä jatkosuuntana, sillä monilla Culinarin asiakkailla oli käytössä iOS:lla toimiva iPad-tabletti. Tästä johtuen kehitystyössä on edetty jättäen mahdollisuudet iOS-version tekemiseen auki niiltä osin, kun ne eivät ole hidastaneet muuta kehitystyötä.

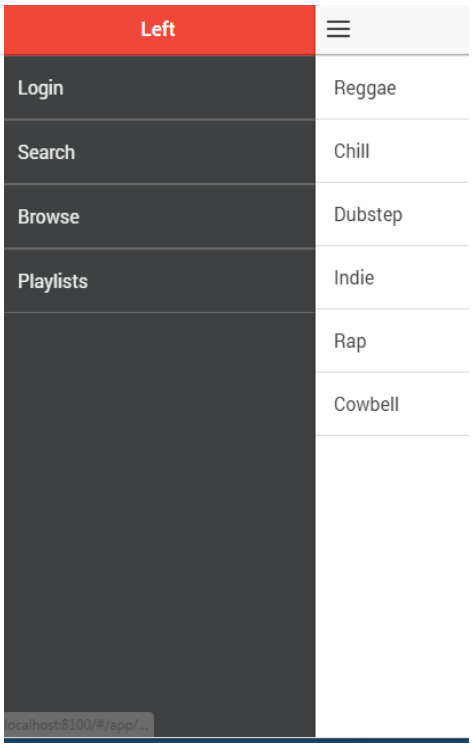
4.1 Porttaus verkkosovelluksesta

Työn ensimmäinen selkeä vaihe oli siirtää olemassa olevat verkkosovelluksen ravintolapään osat hybridisovellukseen. Tällaista alustalta toiseen ohjelman siirtämistä kutsutaan porttaamiseksi.[21] Riippuen ohjelman lähtökohdista, porttaaminen voi olla raskas ja aikaa vaativa prosessi. Esimerkiksi Android sovelluksen porttaaminen iOS sovellukseksi vaatii käytännössä koko ohjelmakoodin uudelleenkirjoittamista, koska kyseiset alustat käyttävät eri ohjelmakieltä ohjelmiensa pyörittämiseen. Ohjelman määrittelyyn käytetty suunnittelutyö on toki uudelleenkäytettävissä porttauksessa, mutta eri alustojen ylläpitäjillä voi olla myös tarkkoja toisistaan poikkeavia vaatimuksia alustalle hyväksyttävälle ohjelmalle. Tämän vuoksi ohjelman määrittelyyn voi joutua tekemään tietylle alustalle suunnattuja muutoksia. Culinarin tapauksessa porttaaminen oli kuitenkin erittäin vaivatonta, koska sen lähtökohtana oli verkkosovellus, joka käytti jo AngularJS:ää ja oli ilmiselvä ratkaisu valita porttaamiseen ohjelmistokehys, joka käytti myös AngularJS:ää. Useat verkkokehittämiseen suunnitellut ohjelmistokehykset tarjoavat nykyään työkaluja hybridisovelluksen kehittämiseen. Tästä esimerkkinä Facebookin kehittämän Reactin hybridisovelluksiin keskittyvä React Native tai Ember.js:lle Poeticin tekemä ember-cli-cordova. Culinarin verkkosovellus olisi siis voitu kehittää jollakin toisella suositulla ohjelmistokehyksellä ja porttaaminen olisi ollut todennäköisesti vaivatonta. Jos lähtökohtana olisi ollut Android sovellus, joka siirretään verkkosovellukseksi, vastaavan kaltaisia työkaluja ei ole olemassa ja koko ohjelma olisi pitänyt kehittää tyhjästä.

4.1.1 Ionic ja testiympäristön vaatimukset

Android sovellusten kehittäminen vaatii muutaman eri työkalun asentamisen, jotta ohjelmakoodin kääntäminen ja testaaminen ovat mahdollista. Kuten aiemmin on jo todettu, Android-sovellukset kehitetään Java kielellä. Vaikka Ionic ja Cordova mahdollistavat kehittämisen verkkosovelluksien kielillä, pinnan alla sovellusta pyöritetään silti Javalla ja Java vaatii Java Development Kitin (JDK). Androidille kehittäminen taas vaatii Androidin kehitystyökalun, Androidin Software Development Kitin (SDK), jonka kanssa Ionic keskustelee Cordovan välityksellä. JDK:n, Android SDK:n, Cordovan ja Ionicin asentamisen jälkeen on mahdollista kääntää tuotettu ohjelma ja testata sen toimivuutta.

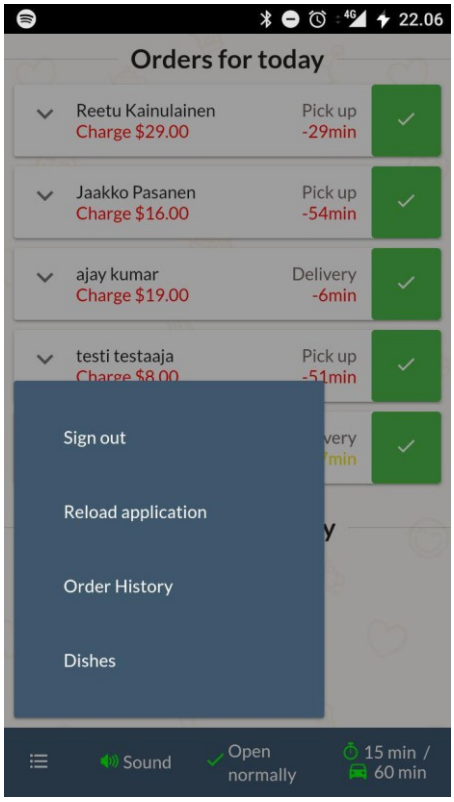
Ionic tarjoaa muutaman erilaisen tavan testata ohjelmaa. Ensimmäinen vaihtoehto on ajaa sovellusta tietokoneella ja testata sitä tietokoneen selaimella. Tämä toimii vain siinä tilanteessa, kun ohjelma ei hyödynnä mobiililaitteen toiminnallisuutta, koska sitä ei selain voi simuloida. Culinarilla tätä vaihtoehtoa käytettiin aluksi vain sen todentamiseen, että sovellus ylipäätään käynnistyy ja toimii. Toisena vaihtoehtona käännetyn ohjelman voi ladata Ionicin pilvipalveluun, josta kehittäjiksi tai testaajiksi määritetyt henkilöt voivat ladata sovelluksen mobiililaitteilleen Ionicin tähän tarkoitukseen kehittämästä sovelluksesta. Vaihtoehto olisi varsin toimiva, jos testaajia olisi useita ja he sijaitsisivat fyysisesti useassa paikassa, mutta yksin sovellusta kehitettäessä sovelluksen lataaminen pilvipalveluun ja sieltä laitteelle jokaisen muutoksen jälkeen kävi aivan liian aikaa vaativaksi prosessiksi. Kolmas vaihtoehto oli asentaa sovellus laitteelle suoraan USB-johtoa pitkin. Tämä oli Culinarin tilanteeseen täydellinen vaihtoehto, sillä yhden



Kuva 4.1 *Ionicin sivuvalikko*

kehittäjän on helppo hallita prosessia ja tehdä muutoksia sovellukseen nopeasti ja testata ne heti. Tässä vaihtoehdossa oli toinenkin hyöty. Kun mobiililaitteesta laitetaan päälle USB-debuggaus moodi, voi tietokoneen selaimessa avata laitteella ajettavan sovelluksen näkymän. Käytännössä tämä mahdollistaa selaimen kehittäjätyökalujen hyödyntämisen, mitä kautta kehittäjä näkee esimerkiksi sovelluksen virhelogit ja renderöidyn HTML-koodin.

Ionic itsessään on varsin yksinkertainen työkalu. Se toimii lähes identtisesti AngularJS:n kanssa, mutta tarjoaa lisäksi työkaluja helpottamaan hybridisovelluksen kehittämistä. Näistä esimerkkinä valmiit visuaaliset elementit, jotka mukailevat yleisiä applikaatioissa hyväksi todettuja elementtejä. Kun hybridisovellusta alettiin kehittää, ensimmäiset sovelluksen versiot rakennettiin näiden valmiiden elementtien avulla. Ohjelmassa oli esimerkiksi monissa muissakin sovelluksissa esiintyvä, kuvassa 4.1 esitetty sivuvalikko, jonka saa auki sivusta pyyhkäisemällä. Tästä huolimatta yhtenevyyden ylläpitämiseksi verkkosovelluksen kanssa, Ionicin omista elementeistä luovuttiin alkukokeilujen jälkeen. Verkkosovelluksen käyttökokemus oli jo itsessään hyvä, eikä pienet parannukset käyttökokemukseen olleet sen lisävaivan arvoisia mitä tulee kahden erilaisen käyttöliittymän ylläpitämisestä. Kuvassa 4.2 on esitettynä Culinarin itse toteuttama valikko.



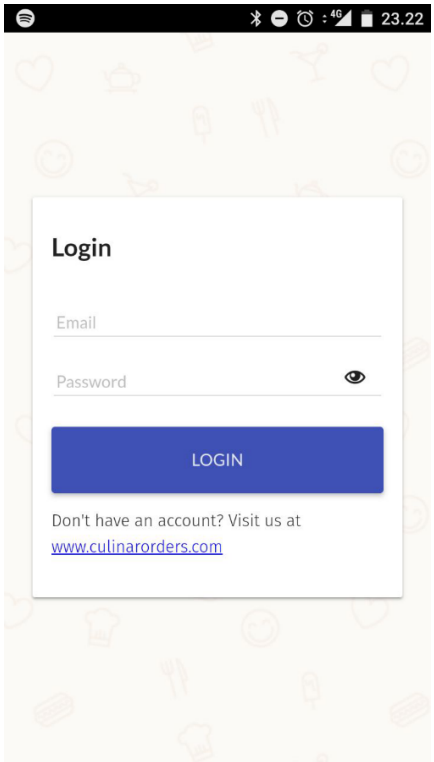
Kuva 4.2 *Culinar sovelluksen valikko*

Testiympäristön valmiiksi saamisen jälkeen ilmeni nopeasti ensimmäinen ongelma, jota ei verkkosovelluksessa ollut. Välillä sovellus ei suostunut alustamaan kaikki muuttujia oikein ohjelmakoodin puolella, jonka vuoksi ohjelma ei päässyt kunnolla lähtemään

käyntiin. Selvisi, että ohjelmakoodi piti jättää odottamaan Ionicin ilmoitusta siitä, että se on suorittanut oman osuutensa ohjelman alustamisesta ennen kuin Culinarin sovellukseen liittyviä alustuksia sai jatkaa. Joissakin tilanteissa Ionic kerkesi kuitenkin suorittamaan omat alustuksensa ennen sovelluksen alustuksia, jonka vuoksi tämä ongelma jäi aluksi huomaamatta. Tähän vaikutti esimerkiksi käytettävän laitteen tyyppi.

4.1.2 Sisäänkirjautuminen

Ensimmäinen selkeä ero verkkosovellukseen, joka vaati enemmän huomiota, oli hybridisovelluksen käynnistäminen. Verkkosovellus tarjoillaan suoraan palvelimelta, jolloin riittää, että osoiteriville kirjoittaa oikean osoitteen, jonka jälkeen selain osaa suorittaa tarjoillun sovelluksen. Kun natiivi sovellus käynnistetään, on tapana näyttää esimerkiksi sovelluksen kehittäjän logo tai muuta informaatiota sovelluksen latautuessa. Tätä käynnistysprosessia varten Ionic tarjosi valmiin työkalun, jolle annettiin kuvat Culinarin logoista. Lopputuloksena oli erikokoisille näytöille skaalaava aloituskuva, joka johtaa ohjelmallisesti määriteltyyn aloitusikkunaan. Verkkosovellukseen pääseminen tapahtuu kirjautumalla www.culinar.fi sivulla, mikä toimi samalla myös kuluttajien sivuna. Hybridisovellus on pelkästään ravintolalle tarkoitettu, jonka vuoksi aloitussivu ei voinut olla sama. Aloitusikkunan päätettiin olevan uusi, vain hybridisovellukselle tehty kirjautumisruutu, jonka näkee kuvassa 4.3.



Kuva 4.3 *Sisäänkirjautumisikkuna*

Verkkosovelluksessa sisään pystyi kirjautumaan sekä asiakkaat, että ravintolat, mutta ravintolan sisään kirjautuessa käyttäjä ohjattiin automaattisesti ravintolan

verkkosovellukseen. Hybridisovelluksessa kirjautuminen taas saa onnistua vain ravintolan tunnuksilla. Culinarin verkkopalvelun rajapintaa tuli siis laajentaa ja alkuperäisen https://culinar.fi/userApi/user/login sisäänkirjautumiseen käytetyn URL:n rinnalle tuli uusi, vain ravintolatunnuksilla toimiva https://culinar.fi/restaurantLogin URL. Uusi URL ei sisällä restaurantApi osaa polussaan, sillä restaurantApiin tehdyt pyynnöt pitää tehdä autentikoituneina, eli ravintolan on pitänyt kirjautua sisään. Tämä käytäntö otettiin käyttöön, koska ravintola rajapintaan ei ole muilla, kuin ravintoloilla asiaa. Ravintola ei voi kuitenkaan vielä olla autentikoitunut palveluun kirjautuessaan, jonka vuoksi rajapintaan piti tehdä poikkeus tässä yhteydessä. Kirjautumissivulle lisättiin myös hyperlinkki Culinarin myyntisivulle, jos sovelluksen sattuisi vahingossa asentamaan joku, jolle palvelua ei ole etukäteen myyty. Uusi käyttäjä ei voi vain luoda itselleen uutta tiliä, koska sovelluksen käyttöönotto vaatii paljon muuta työtä itse tilin luomisen lisäksi. Pelkällä sisäänkirjautumistilillä ravintola ei voisi edes testata järjestelmää, koska Culinarin työntekijän pitää luoda menu, jolta asiakkaat voivat tehdä tilauksen sovellukseen. Uuden tilin luomiseen tarkoitetun sivun tekeminen olisi siis ollut vain kehittäjien ajan huonoa käyttämistä.

4.1.3 Uusi palvelinohjelman rajapinta

Tässä vaiheessa tuli ajankohtaiseksi toteuttaa ja ottaa käyttöön uusi palvelinohjelman rajapinta. Verkkosovelluksen ohjelmakoodissa kaikki HTTP-pyynnöt tehtiin Culinarin palvelimelle vanhan, kaoottisen rajapinnan mukaisesti, jota ei enää ollut tarkoitus käyttää. Määrittelyn yhteydessä jo kerrottiin, että tämä tarkoittaa palvelinohjelman näkökulmasta lähinnä moduulien uudelleenjärjestämistä. Käytännössä vanhassa järjestelmässä kaikki rajapintakutsut oli määritelty yhdessä reittitiedostossa, routes.js:ssä. Moduulia, joka reitittää sisään tulevia pyyntöjä niitä käsittelevälle toiminnallisuudelle kutsutaan hyvän ohjelmointitavan mukaisesti routeriksi. Routerin ei kuitenkaan tarvitse olla yksi massiivinen tiedosto, vaan moduulia voidaan pilkkoa pienempiin, loogisiin kokonaisuuksiin. Routes.js-tiedosto jaettiin uuden skeeman mukaisesti neljään kokonaisuuteen: Jokaista rajapintaa varten luotiin oma tiedostonsa ja lisäksi tehtiin neljäs tiedosto, johon kerättiin kutsut, jotka eivät suoraan kuuluneet minkään rajapinnan alle. Esimerkkinä tällaisesta kutsusta on logiviestien tallentaminen, sillä logiviestit voivat käsitellä mitä tahansa asiaa asiakaspuolen ongelmista ravintolapuolen ongelmiin. Rajapintojen reittitiedostot nimettiin rajapintojensa mukaisesti ja ravintolarajapintaa vastaavan reitityksen nimeksi tuli restaurantApi.js.

HTTP-pyyntöjen reitittäminen toimii seuraavasti: Node.js:llä käynnistettävä palvelinohjelma ensitöikseen käynnistää Express sovelluksen, joka alkaa kuunnella palvelimelle saapuvia HTTP-pyyntöjä. Expressille määritellään myös mihin porttiin saapuvat pyynnöt se ottaa kiinni. Nyt jos palvelin sallii ulkopuoliset yhteydenotot tähän porttiin, voi kuka tahansa Culinarin domain nimen tunteva tehdä HTTP-pyynnön palvelimelle ja Express huolehtii siitä, että tämä pyyntö voidaan käsitellä sovelluksessa.

Kun rajapinnan URL:ä ei ole vielä määritelty reittitiedostoissa, palauttaa sovellus HTTP-virheen koodilla 404, koska Express ei tiedä miten HTTP-pyyntö kyseisellä URL:llä tulee käsitellä. Reittitiedostossa rajapintakutsu määritellään muodossa app.[metodi]([polku], [funktio])(https://expressjs.com/en/4x/api.html). Esimerkiksi tämä voi olla app.get('/restaurantApi/dish/read', dish.read). Nyt Express sovellus vastaanottaessaan HTTP GET-pyynnön ohjaa sen dish moduulin read funktiolle. Express kutsuu tätä funktiota suoritettavaksi vasta kun se on suorittanut omat pinnanalaiset toimintonsa ja parsinut HTTP-kutsusta oleelliset tiedot. Express kutsuu sille annettua funktiota aina kolmella parametrilla, jotka kulkevat yleisesti nimillä request, response ja next. Request parametrissa kulkee asiakasohjelmalta tulevan pyynnön tiedot, kuten GET-pyynnön kysely-parametrit tai POST-pyynnön rungossa kulkevat parametrit. Näin dish moduulin read funktio pystyy suorittamaan pyynnön vaatimat asiat, sillä se saa request parametrin kautta kaiken tarpeellisen informaation. Response parametrin avulla palvelin lähettää asiakasohjelmalle vastauksen, johon sisällytetään tieto vastauksen HTTP-koodista ja mahdollisesta palautettavasta datasta. HTTP-pyynnölle voi määrittää putkeen useamman kutsuttavan funktion, jolloin seuraavaa seuraava funktio kutsutaan suoritettavaksi vasta kun edellisen funktion next parametri laukaistaan. Tällä tavalla reittitiedostoon määriteltiin kaikki uuden rajapinnan mukaiset kutsut ja ohjattiin ne niihin kuuluville moduuleille.

Lopuksi hybridisovelluksen puolella kaikki Culinarin palvelimelle lähtevät kutsut piti muuttaa vastaamaan uutta ravintolarajapintaa. Isossa osassa tapauksia tämä tarkoitti vain URL:n muuttamista, mutta oli myös tilanteita, joissa palvelimelle lähteviä parametreja piti muokata. Kokonaisuudessaan uuden rajapinnan käyttöönottaminen oli selkeä ja suoraviivainen tehtävä, sillä rajapinnan suunnittelu oli hoidettu hyvin, eikä se jättänyt varaa ongelmille yhtä kompastuskiveä lukuun ottamatta.

Ensimmäiset testiympäristöstä tehdyt pyynnöt palvelimelle epäonnistuivat ja syy tälle oli CORS:n puute. Verkkosovelluksen yhteydessä CORS oli asia, jota ei tarvinnut edes pohtia, sillä pyynnöt palvelimelle tehtiin aina samaan alkuperään. Ensimmäisten testien yhteydessä pyyntöjen alkuperä oli kuitenkin eri, sillä sovellus tarjoiltiin ensin paikallisesti ajettavalta palvelimelta, jonka jälkeen Culinarin palvelimelle tehdyt pyynnöt tehtiin tästä paikallisesta palvelimesta poikkeavaan alkuperään. Oletusarvona palvelimet hylkäävät tietoturvan vuoksi kaikki pyynnöt, jotka tulevat ulkopuolisesta alkuperästä. Palvelimella piti laittaa CORS päälle testiympäristöltä tuleville pyynnöille, jolloin pyynnöt tulivat taas normaalisti läpi. Tämä ei kuitenkaan ollut ongelma enää, kun hybridisovellusta testattiin mobiililaitteelta, jolla saman alkuperän käytäntöön liittyen nousi uusi ongelma. Cordovan dokumentaatio kertoo, että oletusarvoisesti pyynnöt sovelluksessa on sallittuja vain file:// skeeman URI:n, joka tarkoittaa, että Cordova sovelluksen alkuperä pyyntöjä muualle tehtäessä on myös file://.[22] File:// skeemalle on kuitenkin annettu suuri vapaus saman alkuperän käytännön suhteen ja on jätetty palvelun toteuttajan vastuulle päättää kuinka toimia.[16] Culinarin verkkopalvelu ei nyt siis tarvitse CORS:a, koska file:// alkuperälle

ei ole määritetty rajoittavaa toimintatapaa, mutta hybridisovellus itsessään ei voi tehdä pyyntöä Culinarin palvelimelle, koska Cordova rajoittaa tämän toiminnan mahdollisten tietoturvaongelmien vuoksi. Tätä varten Cordova on tehnyt liitännäisen nimeltä cordova-plugin-whitelist.[22] Sovellukselle määritellään Sisällön Turvallisuus Menettelytapa (englanniksi Content Security Policy) metatieto juuritason HTML-tiedostoon, jossa sovellukselle kerrotaan mihin ulkopuolisiin verkkoihin HTTP-pyyntöjä saa tehdä. Tässä vaiheessa sovellukselle annettiin oikeudet tehdä pyyntöjä mihin tahansa ulkopuoliseen verkkoon. Tämä ei ole tietoturvan näkökulmasta paras mahdollinen menettelytapa, ja oli tarkoitus muuttaa projektin edetessä, mutta jäi kuitenkin muiden ominaisuuksien korkeamman prioriteetin vuoksi jälkeenpäin vaihtamatta tiukemmaksi.

4.1.4 Sovelluskauppa

Näillä muutoksilla sovellus oli mahdollista käynnistää tabletilla, jos se asennettiin tabletille USB-johdon välityksellä. Jotta sovellus saataisiin yleiseen levitykseen ja ravintoloiden saataville, piti sovellus saada Android alustan Play-sovelluskauppaan. Googlen ja Play-kaupan vaatimukset sovellukselle osoittautuivat erittäin salliviksi, ja koska Culinarin sovelluksessa ei ollut sisäänrakennettuja maksullisia ominaisuuksia tai mitään yleisesti sopimatonta materiaalia, kelpasi se sellaisenaan sovelluskauppaan sisältönsä puolesta heti. Kauppa selvitti tämän sopivuuden kehittäjien sivulla sijaitsevalla kyselyllä. Epäselväksi jäi, tarkastivatko kaupan ylläpitäjät vielä kyselyn paikkansapitävyyden tutkimalla sovellusta tarkemmin, mutta jos näin oli, se ei aiheuttanut lisätoimia Culinarille. Lisäksi kauppaan piti lisätä tiedot sovelluksesta ja sovelluksen nimeksi tuli yksinkertaisesti Culinar For Restaurants.

Viimeisenä vaiheena Ionic projektista piti luoda sovelluskauppaan ladattava APK tiedosto. APK (Android Package Kit) on Androidilla toimivien sovellusten tietotyyppi. Projektin tiedostoista löytyy config.xml niminen tiedosto, jossa on kerrottuna ohjeet Cordovalle APK:n muodostamisesta. Culinarin sovelluksen kannalta oleellisin tieto config.xml:ssä on ohjelman versionumero, ja tieto siitä, mitä Androidin versioita sovellus tukee. Config.xml:stä asetetaan myös mobiililaitteessa näkyvän pikakuvakkeen visuaalinen ulkoasu ja sovelluksen nimi, joka pikakuvakkeessa näkyy. Pikakuvakkeen nimeksi valittiin yksinkertaisesti Culinar. Cordova muodostaa siis config.xml:n ja varsinaisen ohjelmakoodin pohjalta allekirjoittamattoman APK-tiedoston. Tämä tiedosto pitää allekirjoittaa salaisella avaimella, jotta Play-kauppa voi varmistua sinne lisättävän APK-tiedoston tulevan oikeasta lähteestä. Avaimen luontiin, ja APK:n allekirjoittamiseen löytyy työkalut valmiiksi JDK:sta. Lopuksi allekirjoitettu APK piti vielä optimoida käyttäen Android SDK:n tarjoamaa zip align työkalua, joka muodosti lopullisen, sovelluskauppaan ladattavan APK-tiedoston. APK:n lataamisesta kesti noin neljästä viiteen tuntia, että ladattu sovellus löytyi sovelluskaupasta. Tämä kesto on myös uusien versioiden lisäämisessä kauppaan, jonka vuoksi muutoksia on mahdotonta saada välittömästi kauppaan, mikäli ongelmia ilmenee.

4.2 Push-ilmoitukset

Verkkosovelluksen valmiiden osien porttaamisen jälkeen alkoi varsinaisten mobiililaitteille suunnattujen toimintojen toteuttaminen. Näistä isoin ja oleellisin on push-ilmoitukset, joiden avulla Culinarin palvelimien ja laitteilla olevien sovellusten välistä informaation välitystä oli tarkoitus lähteä parantamaan. Käytännössä tämä tarkoittaa tilausten lähettämistä ja laitteiden tilan valvomista.

Push-ilmoitusten toteuttaminen Android ja iOS alustalle on varsin samankaltainen prosessi, vaikka molempien alustojen push-ilmoituspalveluilla onkin omat huomioon otettavat erikoisuudet. Tätä samankaltaisuutta edesauttaa Cordovan valmis työkalu, joka abstrahoi GCM:n ja APNs:n eroja niin paljon kuin mahdollista saman rajapinnan alle. Tämä oli Culinarille merkityksellistä iOS jatkokehitysmahdollisuuden vuoksi, mutta APNs:n erikoisuuksiin ei kiinnitetä tämän enempää huomiota, koska APNs ilmoituksia ei työssä toteutettu. Push-ilmoituksissa Culinarin palvelimella ja mobiililaitteen asiakasohjelmalla on oma roolinsa ja selvyyden vuoksi ensin käydään läpi asiakasohjelman osuus ja sen jälkeen palvelimen rooli.

4.2.1 Asiakasohjelma push-ilmoituksissa

Push-ilmoitusten prosessi alkaa siitä, että asiakasohjelma rekisteröi itsensä GCM-palveluun. Rekisteröinti ilmoittaa GCM:lle asiakasohjelman halun vastaanottaa ilmoituksia, johon GCM vastaa palauttamalla tunnisteen, jolla varsinaisia ilmoituksia voidaan mobiililaitteelle lähettää. Jotta rekisteröityminen on ylipäätään mahdollista, pitää Googlelle luoda kehittäjätili. Kehittäjätilin alle voi luoda projekteja, joiden avulla kyseiset projektit pääsevät käsiksi muun muassa Googlen eri rajapintoihin. Projekti on oleellinen myös GCM:n kautta kulkevissa ilmoituksissa, koska GCM:lle pitää kertoa miltä projektilta rekisteröityvä mobiililaite on vastaanottamassa push-ilmoituksia. Kuten monissa muissa Ioniciin ja hybridisovelluksiin liittyvissä ongelmissa, tarjoaa Cordova rekisteröitymiseen valmiin liitännäisen. Liitännäiselle kerrotaan aiemmin mainittu Googleen rekisteröidyn projektin id-numero, jonka jälkeen liitännäinen pitää huolen siitä, että mobiililaitteen tiedot päätyvät GCM:n. Tunnisteen palauttamista varten ohjelmakoodiin pitää luoda tapahtumakuuntelija, jonka liitännäinen laukaisee saadessaan paluuviestin GCM:ltä. Tapahtumakuuntelija vastaanottaa tunnisteen, jonka se lähettää eteenpäin Culinarin palvelimelle, joka pitää huolen sen tallettamisesta.

Rekisteröitymisen voi suorittaa niin monta kertaa kuin haluaa, jonka vuoksi ei ole vaarallista, vaikka paluuviestissä tulevaa tunnistetta ei ottaisikaan heti talteen. Toimintavarmuuden kasvattamiseksi ja epämääräisten virhetilanteiden välttämiseksi, rekisteröityminen suoritetaan uudestaan neljässä eri ohjelmallisessa vaiheessa.

1. Kun käyttäjä yrittää kirjautua sovellukseen, suoritetaan GCM:n rekisteröityminen. Culinarin palvelimelle ei tehdä kirjautumista ennen kuin

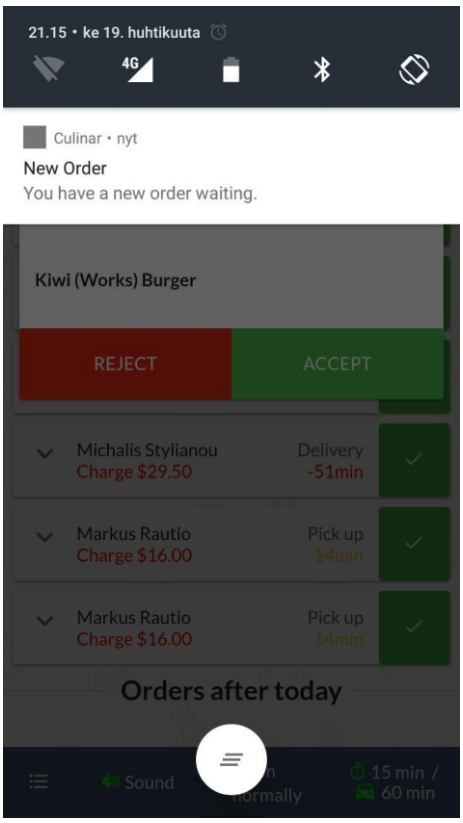
GCM:ltä saadaan paluuviestinä tunniste. Mikäli tunnistetta ei ole saatu kymmenessä sekunnissa, näytetään käyttäjälle virheilmoitus ja kirjautuminen pysäytetään. Jos rekisteröityminen onnistuu, lähetetään tunniste Culinarin palvelimelle sisäänkirjautumisen yhteydessä.

2. Jos sovellus on suljettu kirjautumatta ulos ja sovellus käynnistetään uudestaan, on kirjautumistiedot tallessa laitteen välimuistissa. Nyt käynnistettäessä sovellus uudestaan ei ole tarvetta kirjautua Culinarin palvelimelle, mutta GCM:n rekisteröityminen suoritetaan ennen kuin käyttäjä ohjataan tilausten vastaanottamissivulle. Jos rekisteröityminen epäonnistuu, näytetään käyttäjälle virheilmoitus eikä käyttäjää päästetä tilaussivulle.
3. Jos laitteelta katkeaa internetyhteys, suoritetaan rekisteröityminen uudestaan, kun internetyhteys palautuu. Tunniste lähetetään palvelimelle rekisteröitymisen onnistuessa.
4. Jos laitteen käyttäjä siirtyy sovelluksesta pois ja jättää sen pyörimään taustalle, suoritetaan rekisteröityminen uusiksi, kun käyttäjä avaa sovelluksen laitteella aktiiviseksi. Tunniste lähetetään palvelimelle rekisteröitymisen onnistuessa.

Näin useassa tilanteessa rekisteröityminen saattaa vaikuttaa ylireagoimiselta. Rekisteröitymisellä ei kuitenkaan ole negatiivisia vaikutuksia ja viestien sovellukselle saapuminen on sovelluksen tärkein ominaisuus, joten on parempi olla täydellisen varma, että tämä ominaisuus toimii. Esimerkiksi sovelluksen kehityksen alkuvaiheessa sisäänkirjautuminen ei jäänyt odottamaan tunnisteen saapumista, vaan tunniste lähetettiin Culinarin palvelimelle erikseen. Tästä kuitenkin seurasi mielenkiintoinen virhetilanne: Jos laite sai yhteyden palvelimeen sisäänkirjautumisen hetkellä, mutta ei tunnistetta lähetettäessä, päätyi laite tilaan, jossa se oli sisäänkirjautuneena, mutta palvelimella ei ollut tunnistetta. Nyt sovellus ulkoisesti vaikutti olevan oikeanlaisessa tilassa mutta yhtäkään tilausta ei voitu lähettää sovellukseen. Tämän vuoksi GCM:n rekisteröityminen ja sisäänkirjautuminen ovat riippuvaisia toisistaan, ja vastaavien ennalta näkemättömien tilanteiden varalta rekisteröityminen päätettiin suorittaa useassa tilanteessa. On myös mahdollista, että jostain syystä Culinarin palvelin kadottaa tunnisteen. Jos rekisteröityminen suoritettaisiin vain sisäänkirjautumisen yhteydessä, vaatisi tunnisteen takaisinsaaminen soittoa ravintolaan. Nyt on kuitenkin suuri todennäköisyys, että rekisteröityminen suoritetaan jossakin välissä uudestaan ja palvelin saa tunnisteen uudelleen tuhlaamatta kenenkään työaikaa.

Tavanomaisesti laitteen saadessa GCM:ltä push-ilmoituksen, näytetään laitteen ilmoituskeskuksessa viesti käyttäjälle ilmoituksen saapumisesta, kuten kuvassa 4.4 näytetään. Kun ilmoitusta painetaan, avataan ilmoitukseen liittyvä sovellus ja sovellukselle välitetään ilmoitukseen liittyvä informaatio. Tämä toimintamalli perustuu kuitenkin siihen ajatukseen, että käyttäjä saa omalla ajallaan reagoida ilmoitukseen. Tilaukseen taas ravintolan työntekijän pitää reagoida välittömästi. Tästä johtuen itse Cordovan push-liitännäisen ohjelmakoodia tuli muokata sopivammaksi

ravintolasovelluksen tarpeisiin. Ohjelmakoodista huomattiin liitännäisen reagoivan kolmeen eri tilanteeseen ilmoituksen saapuessa: sovelluksen ollessa aktiivisena, sovelluksen ollessa laitteella taustalla epäaktiivisena ja sovelluksen ollessa kokonaan suljettuna. Liitännäinen muokattiin saavuttamaan jokaisessa tilanteessa sama lopputulos: Ravintolasovellus on aktiivisena laitteella ja ilmoittaa ravintolalle tilauksen saapuneen, riippumatta siitä painaako käyttäjä ilmoituskeskukseen ilmestyvää ilmoitusta. Ensimmäisessä tilanteessa sovellukselle tarvitsee vain ilmoittaa tilauksen saapuneen, toisessa tilanteessa sovellus pitää aukaista aktiiviseksi ja kolmannessa tilanteessa sovellus tulee käynnistää. Sivuoireena tästä seuraa se, että jos käyttäjä haluaisi tehdä laitteella jotain muuta ilmoituksen saapuessa, keskeytyy tämä muu tekeminen ja Culinarin sovellus ponnahtaa auki. Culinarin näkökulmasta tämä on toivottua, koska tilaus ei saa jäädä roikkumaan, mutta käyttökokemus on hieman kehno.



Kuva 4.4 *Tilausviesti ilmoituskeskuksessa*

Alkuvaiheessa push-ilmoituksen rungossa kulki tieto tilauksen sisällöstä. Ilmoituksen rungossa voi kuitenkin kulkea vain kaksi kilotavua informaatiota[23] ja tilauksen sisältäessä paljon ruoka-annoksia, saattoi kaksi kilotavua tulla helposti täyteen. Tämän seurauksena informaatiota katosi matkan varrella. Toinen ongelmakohta muodostui aikaisemmin mainitusta tilanteesta, jossa ohjelma on suljettuna ja pitää käynnistää kokonaisuudessaan ilmoituksen saapuessa. Tätä tilannetta testatessa huomattiin, että liitännäinen ei voi siirtää sovellukselle ilmoituksen mukana tulevaa informaatiota, koska sovellus ei ole käynnissä vastaanottamassa sitä. Ongelmien ratkaisu oli kuitenkin yksinkertainen: Aina kun sovellus saa tiedon ilmoituksen saapumisesta, palaa aktiiviseksi

sovellukseksi laitteella tai käynnistyy, hakee se kaikki tilaukset Culinarin palvelimelta. Koska tilauksissa kulkee tieto niiden tilasta, voi sovellus näyttää ravintolalle nyt kaikki uudet tilaukset, jotka pitää hyväksyä.

4.2.2 Palvelin push-ilmoituksissa

Palvelimen rooli push-ilmoituksissa on lähettää GCM:lle ilmoitus, jonka GCM ohjaa oikealle mobiililaitteelle. Tätä varten palvelimen tulee tallentaa sovellukselta saatu tunniste, jonka avulla GCM tietää mistä laitteesta on kyse. Koska push-ilmoituksien toteutusvaiheessa kysymys APNs:n käyttämisestä oli vielä avoin, tehtiin ratkaisusta myös helposti APNs:n laajennettava. Ratkaisussa päädyttiin hyödyntämään valmista node.js:llä toteutettua Push Server palvelua, joka yhdisti sekä GCM:n, että APNs:n toiminnallisuuden yhteisen HTTP-rajapinnan taakse.[24] Palvelua varten otettiin käyttöön uusi virtuaalipalvelin, minne palvelun HTTP-pyynnöt tehdään. Palvelulla on kaksi tehtävää: ylläpitää tietoa GCM-tunnisteista ja lähettää ilmoitukset GCM:lle. Culinarin palvelimen saadessa sovellukselta tunnisteen, lähettää se tunnisteen Push Server palvelulle HTTP-pyyntönä, mikäli tunnisteen toimittanut sovellukselta saapunut HTTP-pyyntö on muodoltaan oikea. GCM-tunnisteen lisäksi pyynnössä kulkee Culinarin tietokannassa käytettävä ravintolan uniikki tunniste, jolla Push Serverille myöhemmin kerrotaan ilmoitusten kohde. Push Server tallentaa tunnisteen omaan tietokantaansa, eikä tunnistetta tarvitse Culinarin palvelimella enää käyttää. Tässä kohti on hyvä huomata, että ravintolalla voi olla käytössään useampi laite, mikä tarkoittaa useampaa GCM-tunnistetta. Tämä ei ole kuitenkaan mikään ongelma, sillä Push Server tallentaa GCM-tunnisteet kyseisen ravintolan tunnisteen alle. Vastaavasti uuden tilauksen luonnin yhteydessä Culinarin palvelin tekee HTTP-pyynnön Push Serverille, jossa kulkee mukana ravintolan tunniste ja lähetettävän ilmoituksen tiedot. Push Server osaa nyt lähettää ravintolan tunnisteella ilmoituksen jokaiselle ravintolan laitteelle.

Hyväksymättömien tilausten protokollan käytännön toteutus on suoraan liitoksissa palvelimeen ja tilanteisiin, joissa se lähettää tilauksen push-ilmoituksena. Edellisessä kappaleessa esitetty määrittely protokollasta käynnistyy palvelimella samalla, kun push-ilmoitus lähetetään laitteelle.

4.3 Laitteiden monitorointi

Sovelluksen ensimmäinen versio oli mahdollista ottaa käyttöön ravintoloissa, kun push-ilmoitukset oli saatu toimimaan. On huomioitava, että edellisessä kappaleessa kerrottu kehitystyö ja ongelmatilanteiden ratkominen tapahtuivat todellisuudessa vaiheittain, ja ensimmäiset versiot sovelluksesta olivat käytössä samanaikaisesti. Sovellus joutui usein tilaan, jossa Culinarin työntekijöiden piti ottaa yhteys ravintolaan ja auttaa heitä saaman sovellus taas toimintakuntoon. Tätä kautta nousi tarve laitteiden ja sovelluksen tilan monitoroinnille.

Verkkosovelluksesta oli jo opittu, että ravintolasovelluksen yleistä internetin toimivuutta tulee tarkkailla. Hybridisovelluksessa tämä tarkkailu toteutettiin push-ilmoitusten avulla. Joka minuutti jokaiselle GCM:n rekisteröityneelle laitteelle lähetetään monitorointi push-ilmoitus, johon laite vastaa lähettämällä palvelimelle paluuviestin HTTP-pyyntönä. Paluuviestin saapuessa palvelin tallentaa tietokantaan ajanhetken, jolloin alkuperäinen viesti on lähetetty, ja paluuviestin saapumisen ajankohdan. Lähtö- ja saapumisajan yhdistämiseksi viesteille annettiin *universaalisti uniikki tunniste* (UUID, engl. Universally Unique Identifier). UUID on 128-bittinen merkkisarja, johon on sisällytetty erilaisin menetelmin määritettyjä osia ainutlaatuisuuden varmistamiseksi.[25] Kun ilmoitus lähtee palvelimelta, UUID lähtee ilmoituksen kuormana GCM:n kautta laitteelle ja palvelimelle tallennetaan välimuistiin samalla UUID:llä merkitty lähtöajankohta. Sovellus saa tämän UUID:n push-ilmoituksesta, ja liittää sen osaksi palvelimelle lähtevää HTTP-pyyntöä. Nyt palvelin voi hakea UUID:n avulla aiemmin tallennetun lähtöajan ja poistaa merkinnän, kun tiedot on kirjoitettu tietokantaan. Yhteysongelmien aikana viestin kulkeminen ei tietenkään ole taattu. GCM:lle lähtevään ilmoitukseen voi määrittää kuinka pitkään GCM yrittää lähettää ilmoitusta laitteelle, mikäli laite ei vastaanota ilmoitusta.[23] Monitorointi-ilmoituksen eliniäksi määritettiin kymmenen minuuttia, jonka jälkeen palvelin tallentaa tietokantaan vain ilmoituksen lähtöajankohdan ilman saapumisajankohtaa. Paluuviestin puuttuminen on myös merkki Culinarin työntekijöille ravintolassa olevista yhteysongelmista. Kymmenen minuuttia päätettiin sopivan pituiseksi ajaksi, sillä ravintolassa voi olla pieniä lyhyempiä katkoja joihin ei kannata reagoida, sillä ne eivät vaikuta tilauspalvelun toimintavarmuuteen.

Lähtökohtaisesti Android laitteeseen saapuvat ilmoitukset ovat hiljaisia, eli ne eivät itsestään tee laitteen ilmoituskeskukseen kuvassa 4.4 näkyvää viestiä, ellei ohjelmallisesti toisin määritetä. Oletusarvona Cordovan push-liitännäinen tekee tämän ilmoituksen, mutta liitännäistä oli helppo muokata niin, että se vain välittää hiljaisesti sovellukselle tiedon monitorointi-ilmoituksen saapumisesta. Tämä monitorointitapa toimi hyvin, kun piti tarkastella passiivisesti laitteen internetyhteyden toimintaa. Jos ravintolaan ilmaantuu yhtäkkinen ongelma ja pitää varmistua onko laitteeseen yhteys, ei ole mahdollista odottaa mahdollisesti kymmentä minuuttia, että monitorointi reagoi yhteyden katkeamiseen. Passiivisen monitoroinnin rinnalle tehtiin vastaavanlainen aktiivinen ominaisuus, jossa laiteelle lähetetään nappia painamalla push-ilmoitus, ja jos laite vastaa HTTP-pyynnöllä, kerrotaan Culinarin työntekijälle yhteyden olevan kunnossa. Tätä tietoa ei tallenneta mihinkään, sillä se toimii vain hetkellisenä tietona ravintolan internetyhteyden tilasta.

Aluksi Culinarin työntekijöille lähti ilmoitus joka kerta, kun ravintolassa huomattiin yhteyden katkenneen tai yhteyden taas palanneen. Yhteysongelmien huomattiin kuitenkin olevan yleisiä, eikä ravintolan huonoon internetyhteyteen voitu vaikuttaa puhelinsoitolla. Tästä johtuen katkoista ilmoittaminen lopetettiin, sillä sen huomattiin vain lisäävän stressiä ilman keinoja reagoida ongelmiin. Monitoroinnista seurasi toinen varsin mielenkiintoinen ongelma. Palvelinsovelluksella alkoi jossain vaiheessa olla ongelmia

suoriutua tehtävistään ja se alkoi kaatuilla hallitsemattomasti. Asiaa tutkiessa huomattiin, että palvelimen CPU-kuorma oli kasvanut tasaisesti päivä päivältä viikkojen ajan, kunnes palvelin ei enää kyennyt prosessoimaan mitään. Ongelman syyksi paljastui valtava monitorointiviestien tallentaminen ja lukeminen tietokannasta. Minuutti minuutilta jokainen tietokantaluku, joka käsitteli monitorointiviestejä, muuttui raskaammaksi ja näitä lukuja tehtiin paljon, koska tietokanta-arkkitehtuuria ei oltu suunniteltu tarpeeksi hyvin. Ratkaisuna monitorointiviestien historiaa päätettiin ylläpitää vain edellisen päivän ajalta, sillä sitä taaemmaksi katsominen ei tuonut mitään lisäarvoa laitteen tilan tarkkailemiseen.

Monitorointia päätettiin laajentaa nopeasti pelkästä internetyhteyden valvomisesta muiden laitteen valvottavissa olevien ominaisuuksien seuraamiseen. Valvottaviksi asioiksi valittiin akun varaustaso, käytössä olevan laitteen malli, käyttöjärjestelmän versio, sovelluksen versio, WLAN:n SSID, WLAN-signaalin voimakkuus, mahdollinen käytössä oleva mobiiliverkko, mobiiliverkon signaalin voimakkuus, tilausten tila ja sovellukseen liittyen onko sovelluksen ääni mykistetty tai soiko se juuri nyt. Tämä informaatio lisättiin jo palvelimelle aikaisemman monitoroinnin yhteydessä toteutettuihin HTTP-pyyntöihin. Tehtäväksi jäi vain tämän tiedon pyytäminen erillisten Cordova liitännäisten rajapintojen kautta.

4.4 Kohdatut ongelmat

Sovelluksen mahdollisimman nopea käyttöönottaminen ravintoloissa osoittautui parhaaksi ja nopeimmaksi tavaksi kehittää sovellusta, sillä ravintoloiden palaute auttoi kohdentamaan kehityksen resurssit tehokkaasti kriittisimpiin ongelmiin. Tästä kuitenkin seurasi, että tuotantoympäristöön päätyi varsinkin alkuvaiheessa paljon virheitä. Monet virheet johtuivat melko harvinaisista tilanteista, joissa kaksi erillistä prosessia tapahtuivat yhtä aikaa ja sotkivat sovelluksen tilan. Laitteelle piti tehdä uusi ominaisuus, jotta laitteet kykenivät palautumaan tämänkaltaisista virhetilanteista. Uuden ominaisuuden avulla Culinarin kehittäjät voivat lähettää push-ilmoituksen laitteelle, jonka vastaanottaessaan se alustaa sovelluksen uusiksi. Tämän avulla sovellukset pystyttiin palauttamaan monesta ongelmatilanteesta etänä ilman, että ravintolan työntekijät huomasivat mitään tapahtuneen.

Monitoroinnin kautta selvisi paljon ongelmakohtia, joiden olemassaolosta aikaisemmin ei välttämättä ollut edes tietoa. Akun varaustason tarkkailu osoittautui erittäin hyväksi toimenpiteeksi, sillä tätä kautta paljastui yksi iso laitteen käytön ongelmakohta: ravintolat antoivat laitteen varaustason pudota nollaan, eivätkä huomanneet laitteen menevän tämän vuoksi pois päältä. Tämän johdosta sovellukseen tehtiin lisäominaisuus, joka näyttää varoitusviestin käyttäjälle, jos laite ei ole kiinnitettynä laturiin. Viestin saa pois vain kytkemällä laitteen lataukseen. Tämä oli myös ratkaisu toiseen laitteisiin liittyvään ongelmaan. Tabletissa oletusarvona on käytössä virransäästöasetus, joka kytkee WLAN:n pois päältä, kun tabletti ei ole aktiivisessa käytössä. Tästä tietenkin aiheutuu

ongelmia tilauksia vastaanotettaessa ja Culinarin työntekijöiden on mahdotonta kontrolloida ravintoloiden tablettien asetuksia. Tästä johtuen on helpompaa muistuttaa ravintoloita pitämään laitetta latauksessa, jolloin WLAN:n toiminnan kannalta haitallinen virransäästöasetus ei pääse aktivoitumaan.

Laitteen mallin ja käyttöjärjestelmän versionumeron seuraamisen tarkoitus oli yrittää yhdistää mahdolliset ongelmatilanteet laitteen tai käyttöjärjestelmän puutteisiin. Laitteen mallin tietäminen ei loppupeleissä tarjonnut valvontaan lisäarvoa, mutta Androidin version tietämisellä oli käytännön merkitys. Aikaisemmassa vaiheessa, kun sovelluksia ensimmäistä kertaa otettiin käyttöön ravintoloissa, ilmeni erään ravintolan kanssa ongelma käynnistää sovellus oikein. Sovellus kyllä aukesi, mutta ei suostunut näyttämään oikeaa sisäänkirjautumissivua, vaan näytti pelkästään rikkinäisiä elementtejä. Selvitettäessä laitteen tietoja kävi ilmi, että kyseisellä laitteella oli käytössä Androidin vanhempi Jelly Bean versio. Ilmeni, että Jelly Beanissä on ongelma Androidin web view:n näyttämisessä. Kyseinen virhe on korjattu myöhemmissä Androidin versioissa, mutta sen vuoksi web view:tä käyttävä Cordova ei voinut alustaa sovellusta oikein. Web view:lle on kuitenkin olemassa kolmannen osapuolen vaihtoehtoinen versio, jossa tämä ongelma on korjattu. Tästä johtuen Culinarin sovelluksesta piti tehdä Play-sovelluskauppaan kaksi erillistä versiota, joista toinen oli kohdennettu vain vanhoille Androidin versioille. Tässä versiossa Androidin valmiin web view:n sijaan oli asennettuna tämä kolmannen osapuolen web view. Android version valvominen kertoi ensimmäistä kertaa selkeästi, että tätä versiota on kanssa ylläpidettävä, sillä osalla asiakkaista oli käytössä vanhempi Android versio.

Culinarin sovelluksen versionumeroa seuraamalla mahdolliset ongelmat voidaan liittää vanhaan versioon, mikäli uudemmissa versioissa kyseinen ongelma on korjattu, jolloin ratkaisu ongelmaan on pyytää ravintolaa päivittämään sovellus. Langattomien verkkojen seuraaminen tarjoaa joitakin työkaluja verkkoyhteysongelmien ratkaisemiseen. Mikäli mobiililaite käyttää WLAN:a ja WLAN-signaalin voimakkuustason todetaan olevan heikko, ravintolalle voi ehdottaa WLAN-tukiaseman siirtämistä ravintolan sisällä sellaiseen paikkaan, missä signaalia saadaan voimakkaammaksi. Kaikki ravintolat eivät käytä omaa WLAN-tukiasemaa, vaan käyttävät viereisen ravintolan kanssa samaa WLAN:a. Joissakin tapauksissa ravintolan sijaitessa ostoskeskuksessa, ravintola käytti ostoskeskuksen yleistä WLAN:a. Tällaisessa tilanteessa signaali oli todella huono ja vaihteli niin paljon, että signaalia oli mahdoton saada sovellukselle kelvolliseksi. Vaihtoehtona on ehdottaa ravintolalle oman WLAN-tukiaseman hankintaa tai vaihtamista 3G-liittymään. 3G-liittymäkin on toki altis huonolle signaalitasolle, mutta sen ollessa ainut vaihtoehto, sitä kannattaa yrittää, jos ravintola haluaa saada palvelun toimimaan. Langattomien verkkojen yhteysongelmat osoittautuivat haastaviksi, koska niiden ratkaiseminen vaatii toimenpiteitä ravintolan puolesta. Käytännössä ravintolan työntekijä on helppo saada käyttämään pari minuuttia aikaansa ongelmien selvittämiseen, mutta niinkin isot toimenpiteet, kuten WLAN-tukiaseman uudelleensijoittaminen tai uuden

3G:n omaavan mobiililaitteen hankkiminen menevät ravintolan prioriteeteissa todella matalalle, eivätkä välttämättä tule koskaan ratkaistuksi.

Push-ilmoitusten toteutuksessa käytetty erillinen Push Server palvelu ei loppupeleissä ollut paras tapa tehdä asiaa. Lähtökohtaisesti ylimääräisen palvelimen käyttäminen monimutkaistaa palvelinarkkitehtuuria ja useampi komponentti tarkoittaa enemmän mahdollisuuksia virheille. Tämä kasvattaa myös koko palvelun pyörittämisen ylläpitokustannuksia lähes turhaan. Ylimääräisten kustannusten välttämiseksi Push Server palvelua olisi voitu käyttää samalta palvelimelta, millä Culinarin node.js sovellusta ajettiin, kunhan Push Serverin olisi alustanut kuuntelemaan HTTP-pyyntöjä eri portista. Tämä ei kuitenkaan poista muita ratkaisun heikkouksia. Laitteiden monitoroinnin vuoksi Push Serverille tehdään jatkuvasti iso määrä HTTP-pyyntöjä ja tämä laitteille menevien ilmoitusten kierrättäminen toisen palvelun kautta on hitaampaa ja raskaampaa, kuin että ilmoitukset olisivat lähteneet GCM:lle suoraan Culinarin kehittämästä node.js sovelluksesta. Culinarin virhelogeista huomataan, että noin kerran päivässä tässä palveluiden välisessä kommunikaatiossa tapahtuu virhe ja monitoroinnin ilmoitukset eivät mene laitteisiin perille. Tämä ei koko palvelun toiminnan kannalta ole suuri ongelma, mutta kertoo arkkitehtuurillisesta ongelmasta. Jos ongelma johtuu GCM:n kykenemättömyydestä vastaanottaa ilmoituksia, ei ongelma ratkeaisi, vaikka ilmoitukset lähtisivät suoraan Culinarin node.js palvelusta. On kuitenkin ongelmallista, että arkkitehtuurin monimutkaisuuden vuoksi on hankala sanoa, epäonnistuuko ilmoitusten lähettäminen GCM:n vai Push Serverin vuoksi. Jos jatkossa tapahtuu isompi ilmoituksien lähettämiseen liittyvä ongelma, on ongelman selvittäminen ja ratkaiseminen hankalaa. Push Serverin toiminnallisuuden toteuttaminen itse olisi toki vaatinut lähtökohtaisesti enemmän kehitystyötä, mutta nämä tunnit olisi jo mahdollisesti saavutettu takaisin, kun huomioi erillisen palvelun ylläpitämiseen kulutetut tunnit ja virhetilanteiden analysointiin ja pohtimiseen käytetyn ajan. Lisäksi näin olisi säästetty rahaa ja mielenrauhaa paremmin toimivan järjestelmän vuoksi.

4.5 Versionhallinta

Kuten aikaisemmin on todettu, sovellus on kokenut paljon muutoksia ensimmäisen ravintolassa käyttöönotetun version jälkeen. Aikaisemmat, verkkosovelluksesta opitut prosessit eivät suoraan päteneet hybridisovelluksen kehitykseen ja monet asiat piti oppia virheiden kautta.

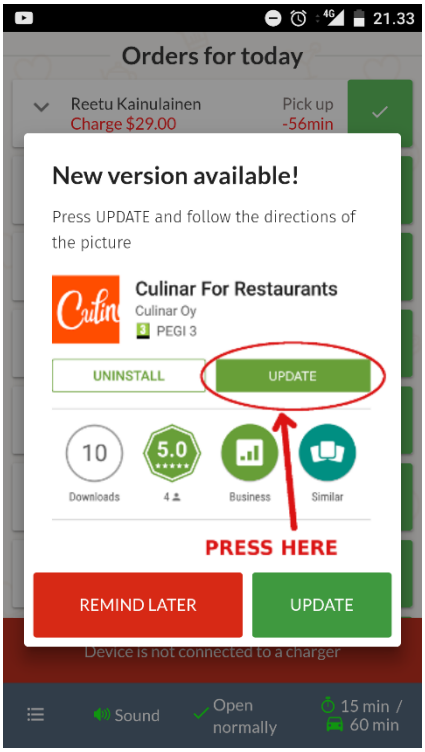
Verkkosovelluksessa uuden version käyttöönottaminen tarkoitti, että tuotantopalvelimelle pusketaan uusi versio sovelluksesta, jonka asiakasohjelmat saavat automaattisesti, kun selain päivittää sivun. Uusi versio oli siis välittömästi saatavilla. Mikäli versiossa havaittiin jokin virhe, se pystyttiin korjaamaan puskemalla korjattu versio välittömästi, tai palaamalla vanhaan versioon, mikäli ongelman korjaamisessa kesti aikaa. Toinen, hienovaraisempi etu oli se, että asiakasohjelma ja selain suoritettiin aina samalla versiolla. Jos rajapintaan piti tehdä muutos, se ei haitannut ohjelman

toimintaa, sillä asiakasohjelmiin päivittyi myös uutta rajapintaa käyttävä versio. Hybridisovelluksen kanssa asiat olivat toisin.

Uuden version puskeminen Play-sovelluskauppaan kestää useamman tunnin ja jos tämä versio sisältää virheitä, ei uutta korjattua versiota, tai vanhaa toimivaa versiota saada sovelluskauppaan useisiin tunteihin. Pahimmassa tapauksessa tämä voi tarkoittaa, että ravintola on kykenemätön vastaanottamaan uusia tilauksia koko sen ajan, kun uuden version puskeminen sovelluskauppaan kestää. Tämä oli odotettavissa oleva ongelma, joka silti päätyi toteutumaan kerran. Onneksi tästä ei seurannut haittoja ravintoloiden toimintaan, sillä ravintolat eivät kerenneet lataamaan tätä versiota, mutta tapaus alleviivasi uusien versioiden testaamisen tärkeyden entistä paremmin. Vaikka tämän ongelmatilanteen pelastukseksi koitui se seikka, että mobiililaitteet eivät välittömästi lataa uusia sovelluksen versioita, tämä sama seikka on myös yksi hybridisovelluksen heikkouksia. Ensinnäkin laitteessa pitää olla päällä asetus uusien versioiden automaattisesta lataamisesta, jota Culinarin on mahdotonta hallita. Toiseksi on täysin mahdoton arvioida missä vaiheessa ajallisesti laite päättää suorittaa tämän lataamisen, sillä se ei tee sitä välittömästi version ollessa saatavilla sovelluskaupasta. Jos uusi sovelluksen versio tuo ravintolan oikeasti tarvitsemia uudistuksia tai korjaa kriittisiä virheitä, ainut tapa varmasti saada uusi versio laitteelle on soittaa ravintolaan ja pyytää heitä manuaalisesti päivittämään sovellus. Tällainen ravintoloiden läpikäyminen on monen tunnin projekti, jolloin sen useasti tekeminen on jo usean miestyöpäivän työ. Tätä varten sovellukseen kehitettiin uusi ominaisuus ratkaisuksi. Kun uusi versio on sovelluskaupassa, palvelimelle kerrotaan uusimman version versionumero. Tämän seurauksena palvelin lähettää kaikille laitteille push-ilmoituksen, jonka saadessaan sovellus näyttää kuvassa 4.5 näkyvän ilmoituksen, joka pyytää ravintolaa päivittämään sovelluksen. Helpottaakseen tätä sovelluksen päivittämistä, ilmoituksessa on hyperlinkki suoraan sovelluksen sivulle Play-sovelluskaupassa. Ilmoituksen saa pienennettyä, että se ei pysäytä tilausten vastaanottamista, mutta se ei katoa ennen kuin ravintola päivittää sovelluksen. Ilmoituksen avulla sovellusten päivittämisestä voidaan varmistua, vaikka ravintolan laitteella ei olisikaan automaattiset sovellusten päivitykset päällä. Toki tässä oli se ongelma, että tämän uuden version saamiseksi ravintoloihin piti ravintolat käydä läpi soittamalla ja pyytää heitä päivittämään sovelluksensa, mutta nyt se piti suorittaa vai kerran.

Aiemmassa kappaleessa mainittiin jo, että vanhoja Android versioita varten sovelluskaupassa on ylläpidettävä kahta erillistä versiota. Käytännössä tämä tehdään kertomalla config.xml:ssä mille Android APK:n versioille kyseinen sovelluksen versio on suunnattu, jonka jälkeen sovelluskauppa osaa tarjota oikealle laitteelle oikean version. Tämän lisäksi kummallekin versiolle tulee antaa oma versio koodinsa config.xml:ssä, jotta uusi lisättävä versio ei yliaja aiemmin lisättyä versiota. Kahden version ylläpitäminen ei varsinaisesti ole hankalaa, sillä versioiden ainut ero config.xml muutosten lisäksi on erillisen web view:n asentaminen vanhemmille Android versioille

suunnattuun sovellukseen. Kolmannen osapuolen web view:n käyttäminen ei kuitenkaan asiakkaiden näkökulmasta ole hyvä asia, sillä se lähes kymmenkertaistaa sovelluksen koon noin 6 megatavusta lähes 50 megatavuun. Mikäli ravintolalla on huono internetyhteys, uusien versioiden lataaminen voi tuottaa hankaluuksia.



Kuva 4.5 *Ilmoitus uudesta versiosta*

Verkkosovellus on kanssa erillinen, ylläpidettävä versio sovelluksesta. Jos sovelluksen käyttöliittymään tehdään muutoksia, tulee nämä muutokset toteuttaa sekä verkkosovellukseen, että hybridisovellukseen. Tämä tuo kehitysprosessiin selkeän ongelmakohdan, sillä Culinarilla näitä muutoksia voi tehdä eri henkilöt, jolloin yksi henkilö mahdollisesti muuttaa ensin verkkosovelluksen käyttöliittymää, jonka jälkeen toisen kehittäjän tehtäväksi jää siirtää nämä muutokset hybridisovellukseen. Verkkosovelluksen ja hybridisovelluksen pariteetin ylläpitämiseen oman haasteensa tuo Ionic, joka pakottaa tietyn version AngularJS:stä. Uuteen AngularJS versioon siirtyminen voi mahdollisesti olla hyödyllistä verkkosovelluksen näkökulmasta, mutta koska se rikkoisi yhtenevyyden hybridisovelluksen kanssa, ei tätä päivitystä voi välttämättä tehdä.

Ravintoloilla voi siis syystä tai toisesta olla käytössä vanha versio sovelluksesta. Jos palvelimen rajapintaan halutaan tehdä muutoksia uusien tarpeiden ilmetessä, nämä muutokset pitää aina toteuttaa hybridisovelluksen kanssa niin, että rajapinnan päivittäminen ei riko ravintoloissa käytössä olevien versioiden toimintaa. Rajapinnan uusiminen sovellusta varten teki rajapinnasta onneksi stabiilin ainakin toistaiseksi. Tulevaisuudessa rajapinnan muutokset voivat vaatia rajapinnan versiointia niin, että vanha rajapinta on vielä käytössä uuden rajapinnan version kanssa, mutta lähitulevaisuudessa rajapinnan uusimiselle ei onneksi ole tarvetta.

4.6 Jatkokehitys

iOS-portin kehittäminen voisi mahdollistaa palvelun käyttöönottamisen ravintoloissa, joissa on jo iOS-laite omistuksessa. Tällaisia pyyntöjä on myynnin yhteydessä tullut mahdollisilta asiakkailta. iOS-portin kehittämisessä on käytännössä kaksi isompaa vaihetta. Push-ilmoitukset on saatava toimimaan Applen APNs:n palvelussa ja sovellus pitää lisätä Applen App Storeen. APNs:n käyttäminen pitäisi olla vaivatonta, sillä työssä käytetyt työkalut tarjoavat tuen ilmoitusten lähettämiseen myös APNs:n kautta. Käytännössä ominaisuutta toteutettaessa voi tietenkin ilmaantua ongelmia, joita ei etukäteen osata odottaa. Sovelluksen lisääminen Applen sovelluskauppaan on huomattavasti työläämpää, kuin sovelluksen lisääminen Play-sovelluskauppaan. Apple vaatii sovellusta täyttämään moniportaisen vaatimuslistan, mihin sisältyy tiukkoja vaatimuksia siitä, millaista sisältöä sovelluksissa saa olla. Ennen kuin sovellus pääsee Applen sovelluskauppaan asti, Applen pitää hyväksyä sovellus sopivaksi, ja hyväksymisprosessi voi kestää viikkoja. iOS-porttiin käytetty kehitystyö ja aika eivät todennäköisesti edistä tarpeeksi Culinarin liiketoimintaa, minkä vuoksi se jää vain yhdeksi tulevaisuuden mahdolliseksi kehitysideaksi.

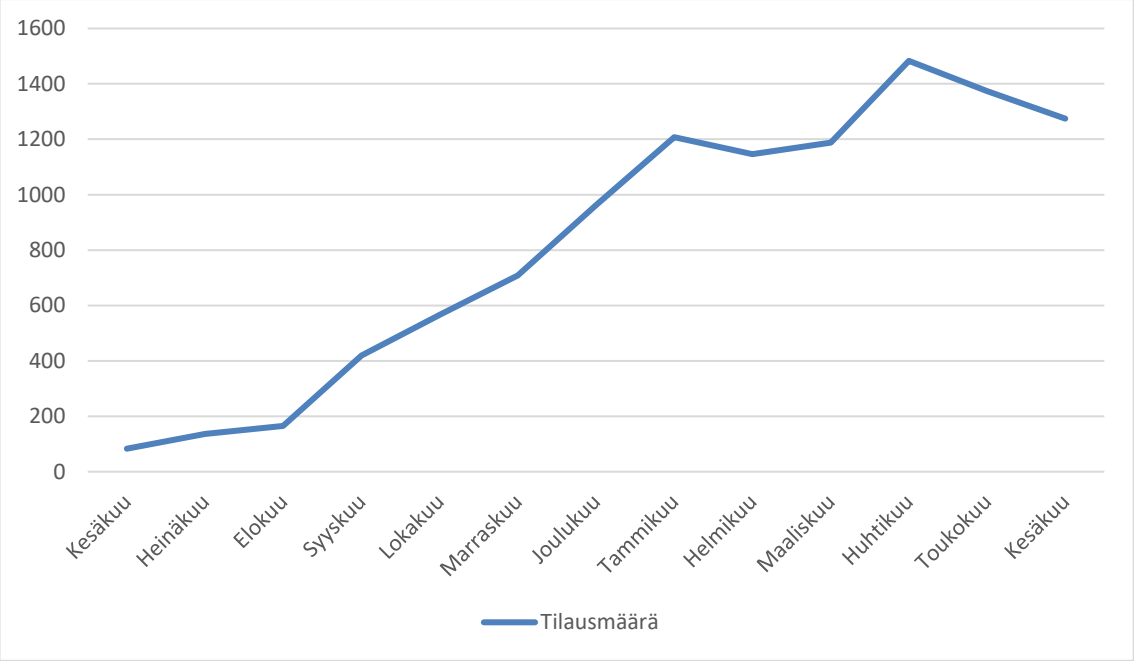
Tilauksen tulostaminen kuittitulostimella on ominaisuus, mikä sopisi esimerkiksi pizzerioihin todella hyvin. Pizzerioissa on pizza-onlinen vuoksi usein valmiina jo kuittitulostin käytössä ja nämä ravintolat ovat tottuneet saamaan verkon kautta tulleet tilaukset tulostettuina. Lisäksi monissa ravintoloissa prosessi etenee niin, että tilauksen jälkeen tilaus kirjoitetaan lapulle ylös, ja lappu toimitetaan keittiöön. Jos sovellus voisi tulostaa tilauksen, tilausta ei tarvitsisi kirjoittaa erikseen ylös. Tällaisen ominaisuuden toteuttaminen vaatii kuitenkin suuren määrän työtä. Eri tulostintyypit toimivat hieman erilaisin tavoin, ja tästä johtuen eri tulostimille pitäisi ohjelmoida omat tulostinrajapintansa. Työn määrä paisuu helposti kohtuuttoman suureksi suhteessa siitä saavutettuun hyötyyn. Tästä johtuen tätä ominaisuutta määriteltiin ja testattiin hyvin pienessä mittakaavassa, mutta jätettiin työpöydälle liian raskaan toteutuksen vuoksi. Jatkossa tätä ominaisuutta voidaan harkita uudelleen, mikäli sen toteuttaminen näyttää vaikuttavan kriittisesti liiketoiminnan jatkamiseen.

5. TULOKSET

Implementaatio -kappaleessa käytiin jo läpi ohjelmistokehityksen prosessista valmistuneita kokonaisuuksia toiminnallisesta näkökulmasta. Tässä kappaleessa keskitytään käsittelemään kvantitatiivisemmin tuotetun sovelluksen kykyä toteuttaa sitä tehtävää, mihin se alun perin on kehitetty.

5.1 Tilausmäärät

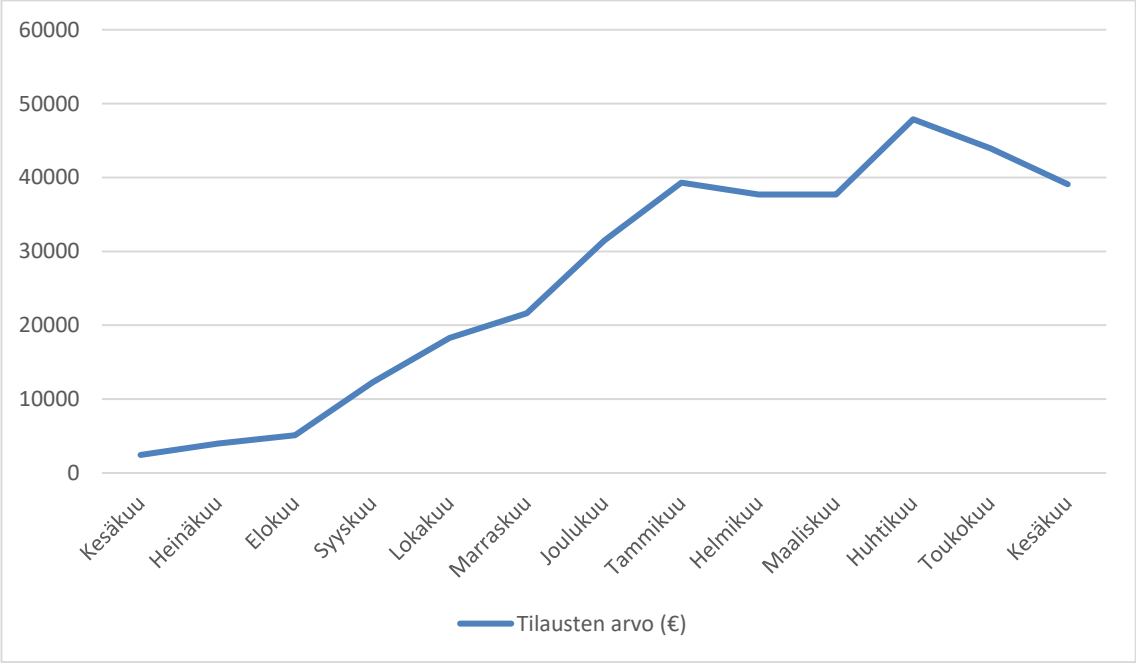
Ravintoloihin saapuvien tilausten määrä on Culinarin liiketoiminnan kannalta tärkein yksittäinen onnistumisen mittari. Tilausmäärä yksinkertaisesti kertoo siitä, että ravintoloiden asiakkaat kokevat palvelun tuovan heille tarpeeksi lisäarvoa, että käyttävät sitä mieluummin kuin muita vaihtoehtoja, esimerkiksi puhelinta. Mikäli ravintola ei saa tilauksia ollenkaan verkkotilauksena, ei ravintolalla ole myöskään syytä maksaa palvelun jatkuvasta käyttämisestä. Tästä johtuen on myös tärkeää nähdä miten ravintoloiden uusi sovellus vaikuttaa tilausmääriin, sillä työ jolla ei ole vaikutusta Culinarin liiketoiminnalliseen onnistumiseen on epäonnistunut.



***Kuva 5.1** Tilausmäärät*

Kuvasta 5.1 nähdään Culinarin kautta kulkeneiden tilausten määrät vuoden 2015 kesäkuusta vuoden 2016 kesäkuuhun asti. Loppukesästä 2015 verkkosovellusta alettiin käyttää tableteilta ja sovellus julkaistiin lokakuun 20. päivä. Tablettien käyttöönottamisen jälkeen tilausten määrä on kasvanut alle 200 tilauksesta kuukaudessa yli 400 tilaukseen kuukaudessa. Tämä tilausten kasvu voidaan selittää sillä, että Culinarin palveluun tuli

uusia ravintoloita, joissa Culinarin konsepti toimi vain tabletin avulla. Sovelluksen käyttöönoton jälkeen tilausmäärät ovat jatkaneet kasvua ja joulukuussa 2015 tilausten määrä oli 962. Taulukosta nähdään myös, että 2016 vuodelle tultaessa tilausmäärät olivat nousseet yli tuhanteen tilaukseen per kuukausi, jossa ne pysyivät sovelluksen aktiivisen kehityksen ajan kesäkuulle 2016. Sovelluksen käyttöönottamisen ja tilausmäärän kasvun välillä on siis selvä korrelaatio. Syksyllä 2015 Culinar myös myi tuotettaan aktiivisesti ravintoloille kahden myyjän voimin, mikä on vaikuttanut tilausmäärän kasvuun. Sovellus on kuitenkin tarjonnut mahdollisuuden toteuttaa tätä myyntiä, sillä ilman sovellusta moni ravintola olisi jäänyt helpommin Culinarin palvelun ulkopuolelle. Sovellukseen on myös tullut kehitystarpeita myynnin kautta. Asiakkaat ovat vaatineet palvelulta tiettyjä toimintoja, minkä vuoksi sovelluksen parantaminen on vaikuttanut edistävästi myyntiin. Tästä johtuen myynnin ja sovelluksen toimivuuden vaikutusta kasvuun on hankala erottaa toisistaan.



***Kuva 5.2** Tilausten kokonaisarvo*

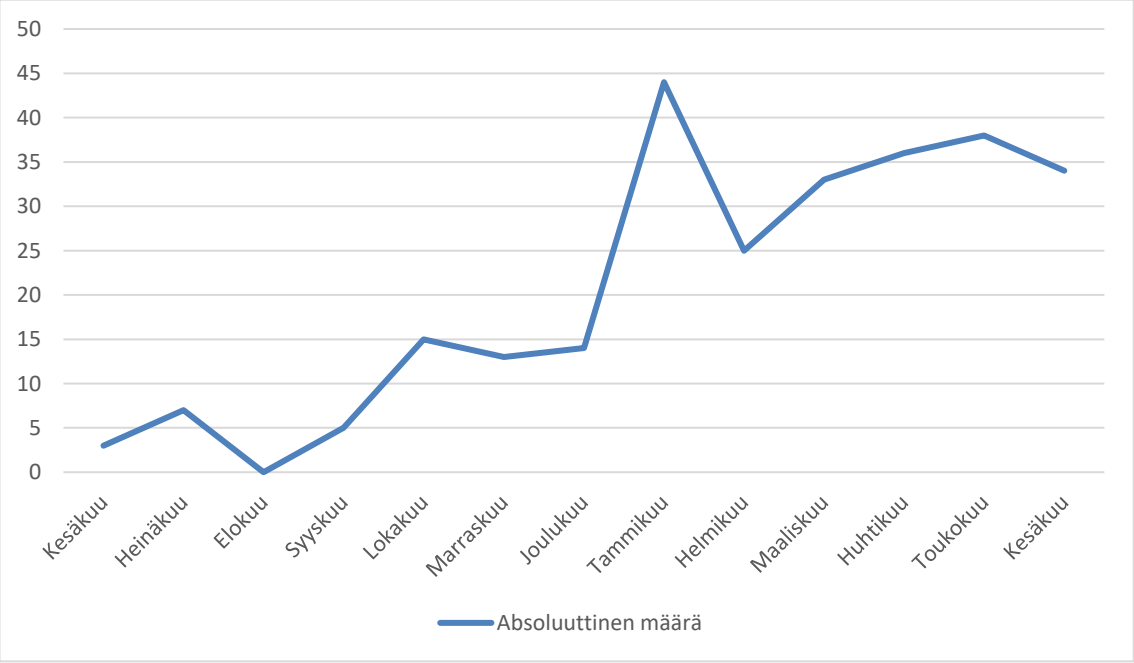
Kuvasta 5.2 näemme vielä, miten tilausmäärät korreloivat ravintoloiden asiakkaiden käyttämään absoluuttiseen rahamäärään. Tilauksen keskimääräinen arvo on heilunut varsin tasaisesti 30-32 euron välissä, jonka vuoksi tilausmäärän ja käytetyn rahamäärän kuvaajat näyttävät lähes identtisiltä. Culinarin kautta kulkeneiden tilausten absoluuttinen rahamäärä on oleellista tiedostaa, sillä siitä saa paremman kuvan sovelluksen arvosta ravintolalle. Mikäli keskiarvotilaus olisi kymmenen euroa pienempi tilausmäärän pysyessä samana, tilanne näyttäisi teknisestä näkökulmasta samalta, mutta ravintolan näkökulmasta mielekkyys käyttää sovellusta laskisi huomattavasti. On kuitenkin mahdotonta arvioida, kuinka suuri liiketoiminnallinen merkitys tällaisilla rahamäärillä on ravintolalle, sillä Culinarilla ei ole tietoa muuta kautta tulevien tilausten määrästä tai arvosta. Tämän lisäksi ei ole tietoa, ovatko Culinarin kautta tulleet tilaukset uusia tilaajia,

vai onko esimerkiksi puhelimen kautta tehdyt tilaukset siirtyneet vain uuteen, digitalisoituneeseen kanavaan. Tarkan määrällisen tiedon puuttuessa on kuitenkin mahdollista arvioida tuotteen mielekkyyttä ravintoloille laadullisesti. Ravintolat, jotka ovat saaneet erittäin vähän, tai ei ollenkaan tilauksia Culinarin kautta, ovat lopettaneet palvelun käyttämisen ennemmin tai myöhemmin. Toisaalta ravintolat, jotka ovat saaneet tasaisesti tilauksia, ovat suullisesti esittäneet tyytyväisyytensä palveluun ja jatkaneet palvelun käyttöä. Vielä oleellisemmin, muutama ravintola, jotka ovat avanneet uuden ravintolan uuteen sijaintiin, ovat ottaneet Culinarin palvelun käyttöön myös uudessa ravintolassa. Tämä kertoo selvästi siitä, että vaikka emme tietäisi palvelun tarkkaa rahallista arvoa ravintolalle, se tuo kuitenkin tarpeeksi lisäarvoa, koska ravintolat haluavat jatkaa ja laajentaa sen käyttämistä.

Vaikka kesäkuun 2016 jälkeen sovellusta ei ole kehitetty ja myyntiä ei ole tehty, tilausmäärät ovat jatkaneet kasvuaan 2016 loppupuolella ja 2017 vuoden alusta. 2017 vuoden maaliskuussa tilauksia tuli ennätyksellisesti 2261 kappaletta, mikä vastaa 79 525 euroa. Sovellus kykenee siis toteuttamaan hyvin tehtäväänsä nykyisessä muodossaan, eikä ravintoloilta ole tullut ylläpidollisia toiveita.

5.2 Hylätyt tilaukset

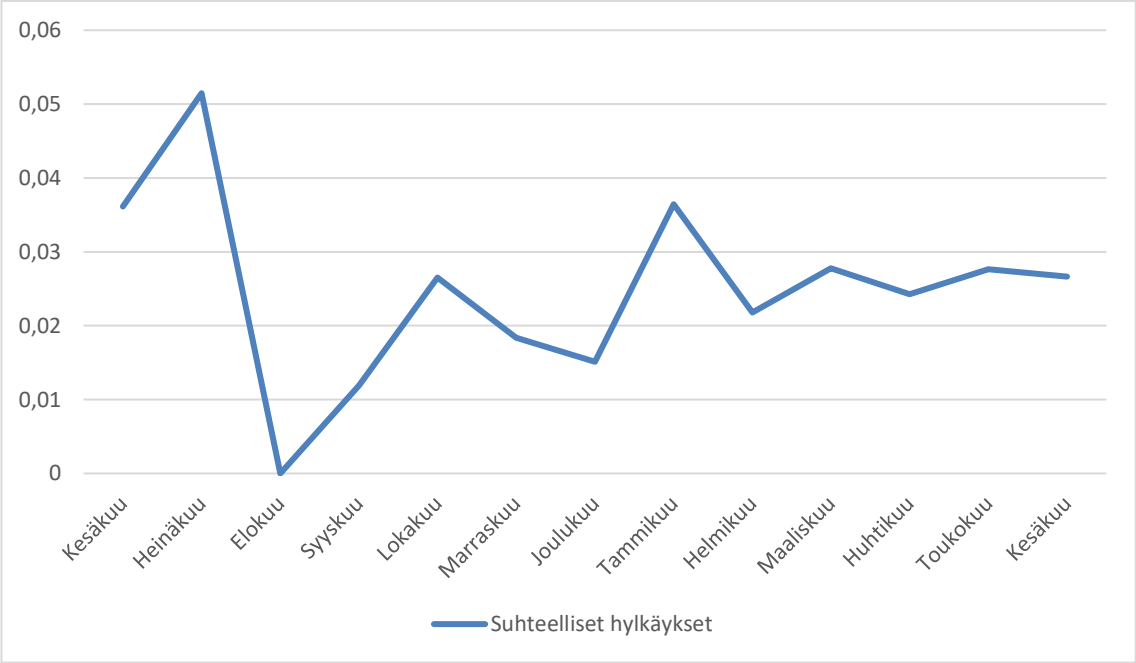
Sovelluksen kykyä toteuttaa tehtäväänsä voidaan arvioida muillakin tavoilla, kuin vastaanotettujen tilausten lukumäärällä. Yksi sovelluksen tehtävistä on huolehtia, että tilaukset tulevat mahdollisimman hyvin perille ravintolaan. Mikäli tilauksia syystä tai toisesta joudutaan hylkäämään, ravintolan asiakkaat voivat saada huonon mielikuvan verkkotilaamisen toimivuudesta eivätkä jatkossa halua enää käyttää Culinarin sovellusta.



Kuva 5.3 *Hylättyjen tilausten kokonaismäärä*

Kuvasta 5.3 näemme hylättyjen tilausten absoluuttisen kokonaismäärän kesäkuiden 2015 ja 2016 välillä.

Hylättyjen tilauksien määrä on kasvanut tasaisesti palvelun olemassaolon ajan, lukuun ottamatta pientä piikkiä tammikuun 2016 aikana. Tasainen hylkäysmäärän kasvu on odotettavissa tilausmäärän kasvaessa, sillä esimerkiksi verkkoyhteyskatkon aikana tulleita tilauksia joudutaan hylkäämään enemmän kuin aiemmin, mikäli katkon aikana on tehty useampi tilaus. Tämän vuoksi absoluuttinen hylkäysmäärä ei vielä riitä kertomaan kunnollista kuvaa hylkäysten kehittymisestä. Kuvasta 5.4 näemme hylättyjen tilausten suhteen hyväksyttyihin tilauksiin. Nyt huomaamme, että suhdeluku on pysynyt keskimääräisesti 0,02:n ja 0,03:n välissä.

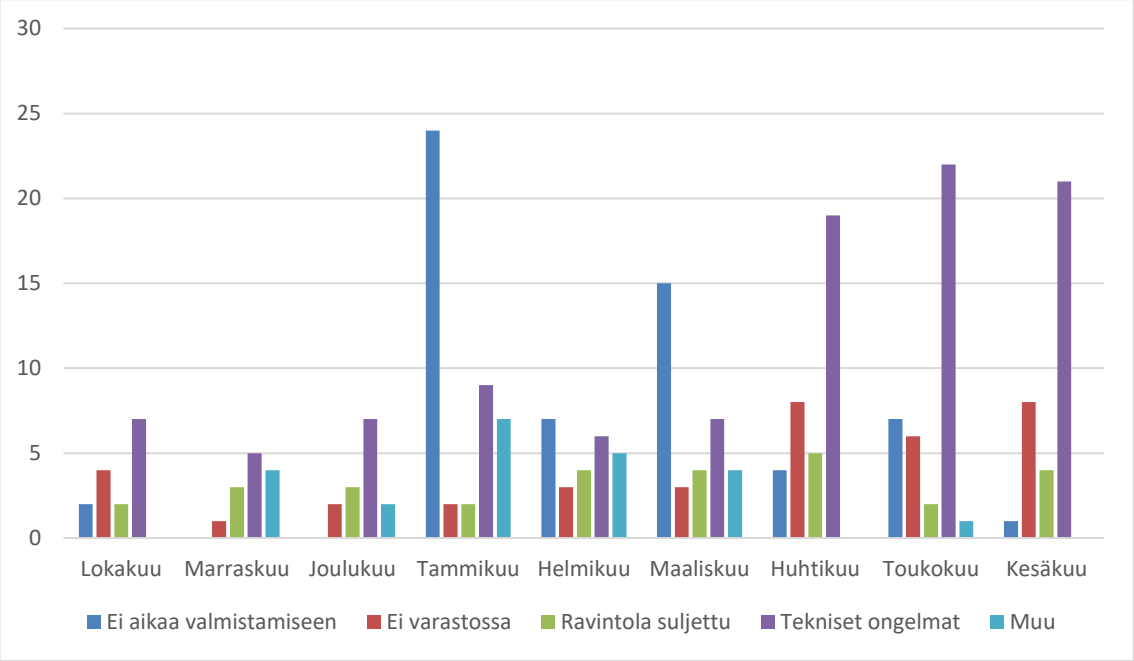


Kuva 5.4 *Suhteellinen tilausten hylkäysmäärä*

Suhdeluvun pysyminen tasaisena kertoo siitä, että siirtyminen tilausten automaattiseen hylkäämiseen 2016 maaliskuussa onnistui sulavasti, eikä aiheuttanut ravintoloiden toimintaan ongelmia. Vaikka tämä siirto ei näkynytkään suoraan tilastoissa, vapautti se kuitenkin enemmän resursseja Culinarin työntekijöiden käyttöön, mikä jatkossa tarkoittaa, että työntekijät voivat keskittyä oleellisempiin työtehtäviin kuin tilausten soittamiseen ravintolaan.

Hylkäysten syistä alettiin tallentaa tietoa lokakuussa 2015. Kuvasta 5.5 näemme miten hylkäykset ovat jakautunut eri aihealueisiin. Kolme ensimmäistä syytä, eli ”ravintolalla ei ole aikaa valmistaa annosta”, ”ravintolalla on pulaa raaka-aineista” ja ”ravintola on suljettu”, olivat Culinarin vaikutusmahdollisuuksien ulkopuolella. 2016 tammikuun piikki kategoriassa ”ei aikaa valmistamiseen” on kuitenkin huomionarvoinen voimakkaassa kasvussaan. Tilausten määrän kasvaessa ravintoloissa on voinut olla

enemmän sovelluksen aiheuttamaa stressiä, mikä on näkynyt ravintolassa kiireellisyytenä, josta on päästy eroon, kun sovellusta on opittu käyttämään paremmin. Tekniset ongelmat kertovat tilanteesta, jossa ravintola ei ole kyennyt hyväksymään tilausta sovelluksesta johtuvista syistä, tai sovellus on hylännyt tilauksen automaattisesti. Muu sarakkeessa suurin osa hylätyistä tilauksista oli Culinarin työntekijöiden hylkäämiä tilauksia, kun Culinar on huomannut tilauksen saapuneen ja yrittänyt soittaa tilauksen ravintolaan, mutta kukaan ei ole vastaanottanut tilausta. Syy on siis voinut olla joko tekninen ongelma, kiire, tai ravintolan kiinnioleminen. Vuoden 2016 huhtikuussa, toukokuussa ja kesäkuussa teknisten ongelmien vuoksi hylättyjen tilausten määrä on kasvanut automaattihylkäyksen käyttöönottamisen vuoksi. Osa näistä tilauksista on kuitenkin aikaisempina kuukausina ollut kategoriassa ”Muu”, jonka vuoksi kyseisen kategorian kasvaminen ei ole niin suurta kuin kuvaaja antaa olettaa.



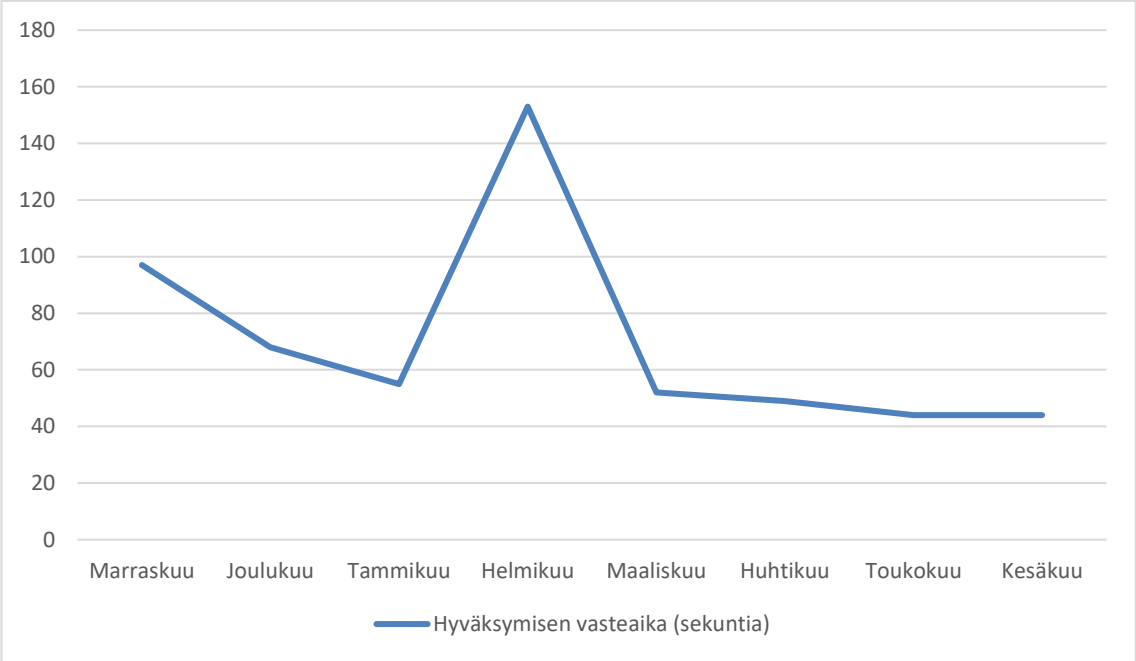
Kuva 5.5 *Hylkäysten syyt*

Hylättyjen tilausten syyt ovat Culinarin sovelluksen osalta jakautuneet odotetusti automaattihylkäykset huomioon otettaessa, ja tässäkin valossa sovellus toimii toivotulla tavalla.

5.3 Tilausten hyväksymisen vasteaika

Tähän asti tarkastellut määreet ovat mitanneet sovelluksen kykyä toimia ravintolan näkökulmasta. Ravintolan asiakkaiden tyytyväisyys palvelun käyttämiseen on kuitenkin yhtä oleellista, sillä jos tilauksia ei palvelun kautta tehdä, ei ravintolalla ole tarvetta sen käyttämiselle. Iso osa ravintolan asiakkaan tyytyväisyydestä muodostuu tilauksen luomisesta, joka suoritetaan Culinarin verkkopalvelun kautta. Ravintolan asiakkaalle näkyvä verkkopalvelu ei kuitenkaan ole osa työtä, jonka vuoksi ravintolan asiakkaan

käyttökokemukseen liittyviä tekijöitä työhön liittyen on huomattavasti vähemmän. Yksi oleellinen tekijä on kuitenkin tilauksen hyväksymisen vasteaika, eli kuinka kauan ravintolan työntekijällä kuluu aikaa hyväksyä tilaus tilauksen luomisajankohdasta laskettuna.



Kuva 5.6 *Hyväksyttyjen tilausten keskimääräinen vasteaika*

Kuvassa 5.6 on esitettynä keskimääräinen hyväksymisen vasteaika sekunteina alkaen marraskuulta 2015, jolloin vasteajan mittaamiseen liittyvää tietoa alettiin kerätä. Vasteajan määrittämiseen on otettu huomioon vain ne tilaukset, jotka ravintolan asiakas on noutanut välittömästi ravintolasta. Tilanteissa, joissa asiakas on tehnyt tilauksen etukäteen, asiakas on voinut tehdä tilauksen ravintolan ollessa kiinni. Tässä tapauksessa ravintolan työntekijä hyväksyy tilauksen vasta ravintolan auetessa, jolloin vasteaika muodostuisi todella suureksi riippumatta siitä, että työntekijä on hyväksynyt sen mahdollisesti hyvinkin nopeasti. Helmikuuta 2016 lukuun ottamatta vasteaika on tasaisesti laskenut 100 sekunnista kohti 40 sekuntia, johon vasteaika näyttäisi tasaantuvan. On hankala arvioida, onko tasainen lasku johtunut sovellukseen tehdyistä parannuksista vai onko ravintolan työntekijät tottuneet käyttämään sovellusta nopeammin. Kyse voi olla myös molemmista.

Helmikuun suuri piikki noin 150 sekuntiin on mielenkiintoinen, mutta todennäköisesti täysin merkityksetön sattuma. Kyse voi olla tuolloin tapahtuneesta ohjelmallisesta virheestä, jonka johdosta vasteajan laskemiseen käytetyt arvot on tallennettu väärin. Toinen mahdollinen selittävä tekijä on se, että tuolloin on sattunut tilauksia, joissa välittömästi noudettava tilaus on syystä tai toisesta jäänyt hyväksymättä ennen ravintolan sulkemisaikaa, tai ravintola on ollut jo kiinni tilauksen mennessä läpi. Näissä tilanteissa kyseiset tilaukset ovat odottaneet pitkän ajan ravintolassa ennen hyväksymistä, ja näin

vasteajan keskiarvo on päässyt kasvamaan suureksi. Yksittäisiä datapisteitä tarkasteltaessa tätä piikkiä ei kuitenkaan näkynyt vasteajoissa, jonka vuoksi on hankala uskoa, että olisi sattunut systemaattista ongelmaa. Ravintoloiden puolelta ei myöskään tullut minkäänlaista tietoa, että tilauksen hyväksymisessä olisi ollut mitään erityistä ongelmaa, joka olisi näkynyt vasteajan kasvamisena.

Vertailukelpoisen datan puuttuessa on hankala arvioida objektiivisesti, onko toukokuun ja kesäkuun arvoksi muodostunut 44 sekuntia hyvä vai huono lopputulos. Vasteaika on kuitenkin parantunut ajan kuluessa, mikä on itsessään hyvä tulos. Subjektiivisesti arvioituna alle minuutin vasteaika ei ole pitkä aika odottaa ravintolan asiakkaalle ja tilausmäärän todistetusti kasvettua tänä aikana, kyseinen vasteaika ei ole ajanut käyttäjiä pois palvelun käyttämisestä.

6. YHTEENVETO

Tulosten valossa sovellus onnistui erinomaisesti teknisen toteutuksensa puolesta. Vaikka sovelluksessa on myös heikkouksia ja sitä voisi kehittää sekä teknisesti paremmaksi, että laajentaa sen toimintakykyä erilaisille laitealustoille, sovellus toteuttaa tehtävää johon se on suunniteltu. Ravintolan asiakkaat pystyvät tekemään tilauksen ja ravintola kykenee tarjoamaan asiakkailleen palvelua nopeasti ja luotettavasti. Tultaessa 2017 vuoden heinäkuulle, sovellus on ollut käytössä kokonaisen vuoden ilman, että sitä on aktiivisesti kehitetty tai ylläpidetty, mikä kertoo hyvästä toimintavarmuudesta. Tilanteista, joissa tilauksen perille meneminen on epävarmaa, kuten verkkoyhteyden katketessa, ei sovellusta kehittämällä voi juurikaan selviytyä paremmin. Sovelluksen tekninen kyky toimia moitteettomasti ei kuitenkaan ole riittävä tapa arvostella työn onnistumista. Sovellus on määritelty ja kehitetty ratkaisemaan ravintoloissa havaittua ongelmaa, mutta ongelma on ensisijaisesti Culinarin itsensä havaitsema. Loppupeleissä verkkotilaamisen puuttuminen ravintoloilta ei ollutkaan niin suuri puute, kuin Culinarilla alun perin sen odotettiin olevan. Tässä tapauksessa siis, vaikka tuote toteutettiin oikein määrittelyn perusteella, määrittely itsessään oli lähtökohtaisesti viallinen. Työssä on keskitytty puhumaan sovelluksen teknisestä puolesta, mutta Culinarin omat motivaatiot sovellukselle on puhtaasti yrityksen liikevaihdon kasvun kehittämisessä ja markkinaosuuden kasvattamisessa. Tästä näkökulmasta sovellusta tarkasteltaessa, se ei tarjoa kuitenkaan tarpeeksi arvoa ravintolalle. Ravintolat ovat kyllä valmiita maksamaan palvelun käyttämisestä, mutta palvelun kehittämisen ja kasvattamisen kustannukset eivät vastaa ravintoloilta saatua tuloa. Tämän vuoksi sovelluksen jatkokehittämiselle ei ole olemassa edellytyksiä, eikä sovellusta enää oteta käyttöön uusiin ravintoloihin, vaikka muutama pyyntö on myöhemmällä ajankohdalla asiasta tullut. Vaikka sovellus epäonnistui tarjoamaan taloudelliset edellytykset liiketoiminnan jatkamiselle, sovelluksen suurin saavutus on sen kehittämisestä opituissa opeissa. Culinar on jatkanut uudelle toimialueelle uudella tuotteella, ja tämä uusi tuote on kehitetty ensisijaisesti ravintolasovelluksen virheet mielessä. Uudella tuotteella tulee olla selkeä asiakaskohderyhmä, joka on valmis maksamaan tuotteesta yrityksen liiketoimintaa ylläpitävän summan. Muussa tapauksessa teknisesti hyvin toimivan tuotteen kehittäminen on kehittäjien ajan hukkaamista. Kasvuyritysten maailmassa epäonnistuminen on vain yksi askel kohti onnistumista.

LÄHTEET

[1] V. Ilmarinen, K. Koskela, Digitalisaatio : yritysjohdon käsikirja, Talentum, Helsinki, 2015, 250 sivua p.

[2] E. Kim, A Secular Shift To Online Food Ordering, web page. Available (accessed 11/22): https://techcrunch.com/2015/05/07/a-secular-shift-to-online-food-ordering/.

[3] D. Sin, E. Lawson, K. Kannoorpatti, Mobile Web Apps - The Non-programmer's Alternative to Native Applications, IEEE, pp. 8-15.

[4] Programming the Mobile Web, 2nd Edition, Ringgold Inc, Beaverton, 2014, .

[5] T. Berners-Lee, Uniform Resource Identifier (URI): Generic Syntax, World Wide Web Consortium, web page. Available (accessed 03/01): http://www.ietf.org/rfc/rfc3986.txt.

[6] M.S. Mikowski, J.C. Powell, Single page web applications, B and W, 2013, .

[7] Y.S. Yilmaz, B.I. Aydin, M. Demirbas, Google cloud messaging (GCM): An evaluation, IEEE, pp. 2807-2812.

[8] I. Warren, A. Meads, S. Srirama, T. Weerasinghe, C. Paniagua, Push Notification Mechanisms for Pervasive Smartphone Applications, IEEE Pervasive Computing, Vol. 13, No. 2, 2014, pp. 61-71.

[9] Google, Google Cloud Messaging, web page. Available (accessed 11/18): https://developers.google.com/cloud-messaging/gcm.

[10] S. Kumari, S.K. Rath, Performance comparison of SOAP and REST based Web Services for Enterprise Application Integration, pp. 1656-1660.

[11] P. Potti, S. Ahuja, K. Umapathy, Z. Prodanoff, Comparing Performance of Web Service Interaction Styles: SOAP vs. REST, Proceedings of the Conference on Information Systems Applied Research, New Orleans Louisiana, USA, pp. 1508.

[12] URI Planning Interest Group, W3C/IETF, URIs, URLs, and URNs: Clarifications and Recommendations 1.0, World Wide Web Consortium, web page. Available (accessed 03/01): https://www.w3.org/TR/uri-clarification/.

[13] R. Fielding, J. Reschke, Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing, Internet Engineering Task Force, web page. Available (accessed 07/17): https://tools.ietf.org/html/rfc7230.

[14] R. Fielding, Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, Internet Engineering Task Force, web page. Available (accessed 07/17): https://tools.ietf.org/html/rfc7231.

[15] L. Dusseault, PATCH Method for HTTP, Internet Engineering Task Force, web page. Available (accessed 07/17): https://tools.ietf.org/html/rfc5789.

[16] A. Barth, The Web Origin Concept, Internet Engineering Task Force, web page. Available (accessed 04/11): https://tools.ietf.org/html/rfc6454.

[17] R. Fielding, Architectural Styles and the Design of Network-based Software Architectures, 2000, Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm.

[18] W.L. Hürsch, C.V. Lopes, Separation of concerns, 1995, .

[19] M. Puska, Langattomat lähiverkot, Talentum, Helsinki, 2005, .

[20] J. Geier, J. Holttinen, Langattomat verkot: perusteet, IT Press, Helsinki, 2005, .

[21] M. Wilcox, L. Aalto, I. Books24x7, Porting to the Symbian platform: open mobile development in C/C++, 1. Aufl.; 1 ed. Wiley, Chichester; Hoboken (NJ), 2009, .

[22] Cordova Plugin Whitelist, Apache Cordova, web page. Available (accessed 04/11): https://cordova.apache.org/docs/en/latest/reference/cordova-plugin-whitelist/.

[23] Google, Messaging Concepts and Options, Google Inc., web page. Available (accessed 04/19): https://developers.google.com/cloud-messaging/concept-options.

[24] S. Basset, Push Server, Smile Mobile Team, web page. Available (accessed 04/20): https://github.com/Smile-SA.

[25] P.J. Leach, M. Mealling, R. Salz, A universally unique identifier (uuid) urn namespace, 2005, .